

LECTURE NOTES IN COMPUTATIONAL SCIENCE AND ENGINEERING 136

Hans-Joachim Bungartz · Severin Reiz
Benjamin Uekermann · Philipp Neumann
Wolfgang E. Nagel *Editors*

Software for Exascale Computing SPPEXA 2016 – 2019

Editorial Board
T. J. Barth
M. Griebel
D. E. Keyes
R. M. Nieminen
D. Roose
T. Schlick

OPEN ACCESS

Springer

www.dbooks.org

Lecture Notes in Computational Science and Engineering **136**

Editors:

Timothy J. Barth
Michael Griebel
David E. Keyes
Risto M. Nieminen
Dirk Roose
Tamar Schlick

More information about this series at http://www.springer.com/series/3527

www.dbooks.org

Hans-Joachim Bungartz • Severin Reiz •
Benjamin Uekermann • Philipp Neumann •
Wolfgang E. Nagel
Editors

Software for Exascale Computing - SPPEXA 2016-2019

Springer Open

Editors
Hans-Joachim Bungartz
Technische Universität München
Garching, Germany

Severin Reiz
Technische Universität München
Garching, Germany

Benjamin Uekermann
Department of Mechanical Engineering
Eindhoven University of Technology
Eindhoven, The Netherlands

Philipp Neumann
Helmut-Schmidt-Universität Hamburg
Hamburg, Germany

Wolfgang E. Nagel
Technische Universität Dresden
Dresden, Germany



ISSN 1439-7358 ISSN 2197-7100 (electronic)
Lecture Notes in Computational Science and Engineering
ISBN 978-3-030-47955-8 ISBN 978-3-030-47956-5 (eBook)
https://doi.org/10.1007/978-3-030-47956-5

Mathematics Subject Classification: 82-08

This book is an open access publication.

© The Editor(s) (if applicable) and The Author(s) 2020
Open Access This book is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.
The images or other third party material in this book are included in the book's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the book's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG.
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

www.dbooks.org

Preface

This volume summarizes the research done and results obtained in the second funding phase of the Priority Program 1648 "Software for Exascale Computing" (SPPEXA) of the German Research Foundation (DFG). In that respect, it both provides an overview of SPPEXA's achievements and represents a continuation of Vol. 113 in Springer's series "Lecture Notes in Computational Science and Engineering", the corresponding report of SPPEXA's first funding phase.

For some general remarks on the uniqueness of SPPEXA—as the first strategic, i.e. board-initiated Priority Program of DFG; as the first tri-national Priority Program with synchronized collaborative research in Germany, France, and Japan; as a multi-disciplinary endeavor involving informatics and mathematics, but also various fields from engineering, the sciences, and the life sciences; and as the first holistic approach to research on High-Performance Computing (HPC) software at the level of fundamental research—we refer to the overview contribution of Bungartz et al. (see chapter "Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA") in this volume. There, also some statistics are provided.

The spirit of the international collaboration, whether in a bi-lateral (German–Japanese) or in a tri-lateral (French–Japanese–German) setting, can be found and felt in several of the reports of 16 out of 17 SPPEXA consortia. This structured and institutionalized collaboration was not easy to establish, and we are grateful for the shared enthusiasm, commitment, and support of the three involved funding agencies: the German Research Foundation (DFG), the Agence Nationale de la Recherche (ANR), and the Japan Science and Technology Agency (JST). The synergies emerging from bringing together the expertise of groups from three countries did not only boost the respective project work itself, it also prepared the ground for ongoing partnerships as well as for a topical extension towards the interplay of HPC and Artificial Intelligence—a field that both benefits tremendously from HPC and, at the same time, fosters HPC with new concepts.

As always, many people helped to make SPPEXA in general and this volume in particular a great success. Concerning the first, our thanks go to the agencies already mentioned and their responsible officers; then to all the SPPEXA researchers

in 17 consortia who made SPPEXA such a wonderful and productive research experience; and finally to all helping hands that supported SPPEXA in terms of organizing and hosting events such as workshops, doctoral retreats, minisymposia, gender workshops, annual plenary meetings, and so forth. Moreover, concerning the preparation of this volume, we are grateful to Dr. Martin Peters and Leonie Kunz from Springer for their support—as in previous cases, it was again a pleasure to collaborate. Finally, we thank Mirco Troue, Tina Angerer, and Michael Obersteiner for their support in proofreading and compiling this book.

The first exascale systems are expected to be available in about one year. For sure, there is still a lot of work to be done to let cutting-edge science applications fully exploit their potential. However, we are fully convinced that SPPEXA contributed significantly to pave the way towards exascale computers and their usage.

Garching, Germany — Hans-Joachim Bungartz
Garching, Germany — Severin Reiz
Eindhoven, Netherlands — Benjamin Uekermann
Hamburg, Germany — Philipp Neumann
Dresden, Germany — Wolfgang E. Nagel

www.dbooks.org

Contents

Part I SPPEXA: The Priority Program

Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA 3
Hans-Joachim Bungartz, Wolfgang E. Nagel, Philipp Neumann, Severin Reiz, and Benjamin Uekermann

A Perspective on the SPPEXA Collaboration from France 19
Nahid Emad

A Perspective on the SPPEXA Collaboration from Japan 23
Takayuki Aoki

Part II SPPEXA Project Consortia Reports

ADA-FS—Advanced Data Placement via Ad hoc File Systems at Extreme Scales 29
Sebastian Oeste, Marc-André Vef, Mehmet Soysal, Wolfgang E. Nagel, André Brinkmann, and Achim Streit

AIMES: Advanced Computation and I/O Methods for Earth-System Simulations 61
Julian Kunkel, Nabeeh Jumah, Anastasiia Novikova, Thomas Ludwig, Hisashi Yashiro, Naoya Maruyama, Mohamed Wahib, and John Thuburn

DASH: Distributed Data Structures and Parallel Algorithms in a Global Address Space 103
Karl Fürlinger, José Gracia, Andreas Knüpfer, Tobias Fuchs, Denis Hünich, Pascal Jungblut, Roger Kowalewski, and Joseph Schuchart

ESSEX: Equipping Sparse Solvers For Exascale 143
Christie L. Alappat, Andreas Alvermann, Achim Basermann, Holger Fehske, Yasunori Futamura, Martin Galgon, Georg Hager, Sarah Huber, Akira Imakura, Masatoshi Kawai, Moritz Kreutzer, Bruno Lang, Kengo Nakajima, Melven Röhrig-Zöllner, Tetsuya Sakurai, Faisal Shahzad, Jonas Thies, and Gerhard Wellein

ExaDG: High-Order Discontinuous Galerkin for the Exa-Scale 189
Daniel Arndt, Niklas Fehn, Guido Kanschat, Katharina Kormann, Martin Kronbichler, Peter Munch, Wolfgang A. Wall, and Julius Witte

Exa-Dune—Flexible PDE Solvers, Numerical Methods and Applications 225
Peter Bastian, Mirco Altenbernd, Nils-Arne Dreier, Christian Engwer, Jorrit Fahlke, René Fritze, Markus Geveler, Dominik Göddeke, Oleg Iliev, Olaf Ippisch, Jan Mohring, Steffen Müthing, Mario Ohlberger, Dirk Ribbrock, Nikolay Shegunov, and Stefan Turek

ExaFSA: Parallel Fluid-Structure-Acoustic Simulation 271
Florian Lindner, Amin Totounferoush, Miriam Mehl, Benjamin Uekermann, Neda Ebrahimi Pour, Verena Krupp, Sabine Roller, Thorsten Reimann, Dörte C. Sternel, Ryusuke Egawa, Hiroyuki Takizawa, and Frédéric Simonis

EXAHD: A Massively Parallel Fault Tolerant Sparse Grid Approach for High-Dimensional Turbulent Plasma Simulations 301
Rafael Lago, Michael Obersteiner, Theresa Pollinger, Johannes Rentrop, Hans-Joachim Bungartz, Tilman Dannert, Michael Griebel, Frank Jenko, and Dirk Pflüger

EXAMAG: Towards Exascale Simulations of the Magnetic Universe 331
Volker Springel, Christian Klingenberg, Rüdiger Pakmor, Thomas Guillet, and Praveen Chandrashekar

EXASTEEL: Towards a Virtual Laboratory for the Multiscale Simulation of Dual-Phase Steel Using High-Performance Computing 351
Axel Klawonn, Martin Lanser, Matthias Uran, Oliver Rheinbach, Stephan Köhler, Jörg Schröder, Lisa Scheunemann, Dominik Brands, Daniel Balzani, Ashutosh Gandhi, Gerhard Wellein, Markus Wittmann, Olaf Schenk, and Radim Janalík

ExaStencils: Advanced Multigrid Solver Generation 405
Christian Lengauer, Sven Apel, Matthias Bolten, Shigeru Chiba, Ulrich Rüde, Jürgen Teich, Armin Größlinger, Frank Hannig, Harald Köstler, Lisa Claus, Alexander Grebhahn, Stefan Groth, Stefan Kronawitter, Sebastian Kuckuk, Hannah Rittich, Christian Schmitt, and Jonas Schmitt

www.dbooks.org

ExtraPeak: Advanced Automatic Performance Modeling for HPC Applications 453
Alexandru Calotoiu, Marcin Copik, Torsten Hoefler, Marcus Ritter, Sergei Shudler, and Felix Wolf

FFMK: A Fast and Fault-Tolerant Microkernel-Based System for Exascale Computing 483
Carsten Weinhold, Adam Lackorzynski, Jan Bierbaum, Martin Küttler, Maksym Planeta, Hannes Weisbach, Matthias Hille, Hermann Härtig, Alexander Margolin, Dror Sharf, Ely Levy, Pavel Gak, Amnon Barak, Masoud Gholami, Florian Schintke, Thorsten Schütt, Alexander Reinefeld, Matthias Lieber, and Wolfgang E. Nagel

GROMEX: A Scalable and Versatile Fast Multipole Method for Biomolecular Simulation 517
Bartosz Kohnke, Thomas R. Ullmann, Andreas Beckmann, Ivo Kabadshow, David Haensel, Laura Morgenstern, Plamen Dobrev, Gerrit Groenhof, Carsten Kutzner, Berk Hess, Holger Dachsel, and Helmut Grubmüller

MYX: Runtime Correctness Analysis for Multi-Level Parallel Programming Paradigms 545
Joachim Protze, Miwako Tsuji, Christian Terboven, Thomas Dufaud, Hitoshi Murai, Serge Petiton, Nahid Emad, Matthias S. Müller, and Taisuke Boku

TerraNeo—Mantle Convection Beyond a Trillion Degrees of Freedom 569
Simon Bauer, Hans-Peter Bunge, Daniel Drzisga, Siavash Ghelichkhan, Markus Huber, Nils Kohl, Marcus Mohr, Ulrich Rüde, Dominik Thönnes, and Barbara Wohlmuth

Acronyms

AABB	Axis-aligned bounding box
AMG	Algebraic multigrid
AoS	Array of structures
AST	Abstract syntax tree
BiCGSTAB	Biconjugate gradient stabilized
CFD	Computational fluid dynamics
CFL	Courant-Friedrichs-Lewy
CG	Conjugate gradient
CGS	Coarse-grid solver
CR	Conjugate residual
CSE	Common subexpression elimination
CST	Concrete syntax tree
CSV	Comma-separated values
DAG	Directed acyclic graph
DG	Discontinuous Galerkin
DSL	Domain-specific language
DTFT	Discrete-time Fourier transform
FAS	Full approximation scheme
FD	Finite differences
FDM	Finite difference method
FE	Finite elements
FEM	Finite element method
FFC	FEniCS form compiler
FFT	Fast Fourier transform
FPGA	Field-programmable gate array
FV	Finite volumes
FVM	Finite volume method
GA	Genetic algorithm
GCC	GNU compiler collection
GMG	Geometric multigrid
GP	Genetic programming

www.dbooks.org

GPL	General-purpose language
HPC	High-performance computing
ICC	Intel C++ compiler
IR	Intermediate representation
ISL	Integer set library
IV	Internal variable
JIT	Just-in-time
LBM	Lattice Boltzmann method
LCCSE	Loop-carried common subexpression elimination
LFA	Local Fourier analysis
MINRES	Minimal residual
MSVC	Microsoft visual C++
PDE	Partial differential equation
PGAS	Partitioned global address space
RBGS	Red-black Gauss-Seidel
RSDFT	Real space density functional theory
SIMPLE	Semi-implicit method for pressure linked equations
SoA	Structure of arrays
SOR	Successive over-relaxation
SPDE	Stochastic partial differential equation
SPL	Software product line
SpMV	Sparse matrix-vector multiplication
SWE	Shallow water equations
TBB	Threading building blocks
TPDL	Target platform description language
TSC	Time stamp counter
TSFC	Two stage form compiler
UFC	Unified form-assembly code
UFL	Unified form language

Part I
SPPEXA: The Priority Program

www.dbooks.org

Software for Exascale Computing: Some Remarks on the Priority Program SPPEXA



Hans-Joachim Bungartz, Wolfgang E. Nagel, Philipp Neumann, Severin Reiz, and Benjamin Uekermann

Abstract SPPEXA, the Priority Program 1648 "Software for Exa-scale Computing" of the German Research Foundation (DFG), was established in 2012. SPPEXA was DFG's first strategic Priority Program—strategic in the sense that it had been the initiative of DFG's board to suggest a larger and trans-disciplinary funding scheme to support the development of software at all levels that would be able to benefit from future exa-scale systems. A proposal had been formulated by a team of scientists representing domains across the STEM fields, evaluated in the standard format for Priority Programs, and financed via special funds. Operations started in January 2013, and after two 3-year funding phases and a cost-neutral extension, SPPEXA's activities will come to an end by end of April, 2020. A final international symposium took place on October 21–23, 2019, in Dresden, and this volume of Springer's Lecture Notes in Computational Science and Engineering—the second SPPEXA-related one after the corresponding report of Phase 1 (see Appendix 3 in [1])—contains reports of 16 out of 17 SPPEXA projects (the project ExaSolvers will deliver its report as a special issue of Springer's journal Computing and Visualization in Science) and is, thus, a comprehensive overview of research within SPPEXA.

While each single project report emphasizes the respective project's individual research outcomes and, thus, provides one perspective of research in SPPEXA, this contribution, co-authored by the two scientific coordinators—Hans-Joachim

H.-J. Bungartz · S. Reiz (✉)
Technical University of Munich, Garching, Germany
e-mail: bungartz@in.tum.de; reiz@in.tum.de

W. E. Nagel
Technical University of Dresden, Dresden, Germany

P. Neumann
Helmut-Schmidt-Universität Hamburg, Hamburg, Germany
e-mail: philipp.neumann@hsu-hh.de

B. Uekermann
Eindhoven University of Technology, Eindhoven, Netherlands

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_1

Bungartz and Wolfgang E. Nagel—and by three of the four researchers that have served as program coordinator over the years—Philipp Neumann, Benjamin Uekermann, and Severin Reiz—emphasizes the program SPPEXA itself. It provides an overview of the design and implementation of SPPEXA, it highlights its accompanying and supporting activities (internationalization, in particular with France and Japan; workshops; doctoral retreats; diversity-related measures), and it provides some statistics. It, thus, complements the papers from SPPEXA's research consortia collected in this volume.

1 Preparation

While super*computers* were recognized early as an important research infrastructure for German science and have been since then on the agenda (recommendations of the German Science Council (Wissenschaftsrat), introduction of the performance pyramid, Gauss Centre for Supercomputing, Gauss Alliance, NHR—Nationales Hochleistungsrechnen), the situation for super*computing* has always been quite different. First, the funds for HPC systems are typically limited to investments, i.e. the machinery; the current NHR initiative takes a more comprehensive view. Second, software development is frequently not considered as "science", which entails that neither typical projects in informatics or mathematics nor their counterparts in fields of application cover more than prototype development. Recently, BMBF's HPC software program and DFG's sustainable scientific software initiative, fortunately, have acknowledged the crucial role of software for HPC and support software development explicitly. Third, HPC software development has happened in Collaborative Research Centers or similar formats before, but mostly in an isolated way: an informatics initiative contained an HPC software project as an application, or a physics initiative contained a simulation- or HPC-oriented project. But all this hardly ever looked at more than one peculiar aspect at a time, and it was at most an interdisciplinary endeavor of two fields.

However, when Moore's law at least gets exhausted a bit and performance gains are more and more achievable through a more and more massive parallelism only, it is obvious that software and its performance and scalability play an increasingly crucial part. Therefore, the challenges at the eve of the exa-scale era required more—and that's actually what happened elsewhere, for example in the U.S. or in Japan: a significant, concerted initiative, bringing together informatics, mathematics, and several domains of application, comprising all relevant aspects of HPC software. That's where SPPEXA entered the stage.

2 Design Principles

SPPEXA was designed to provide a holistic approach to HPC software, comprising the aspects most relevant for ensuring the efficient use of current and upcoming high-end supercomputers, and to do this via exploring both evolutionary and

www.dbooks.org

disruptive research threads. Six research directions were identified as crucial ones: (1) Computational Algorithms, (2) Application Software, (3) System Software and Runtime Libraries, (4) Programming, (5) Software Tools, and (6) Data Management. Computational algorithms, such as fast linear solvers or eigensolvers, are a core numerical component of many large-scale application codes—both classical simulation-driven and recent data analytics-oriented ones. If scalability cannot be ensured here, the battle is already almost lost. Application software is the "user" of HPC systems, typically appearing as legacy codes that have been developed over many years. Increasing their performance via a co-design that addresses both the "systems—algorithms" and the "algorithms—applications/models" interfaces and combines algorithm and performance engineering is vital. Performance engineering can't succeed without progress in compilers, monitoring, code optimization, verification support, and parallelization support (such as auto-tuning)—which underlines the importance of system software and runtime libraries as well as of tools. Programming, including programming models, is probably the topic where the need for a balance of evolutionary research (improve and extend existing programming models, e.g.) and revolutionary approaches (explore new programming models, new language concepts such as Domain-Specific Languages) gets most obvious. Data management, finally, has always been HPC-relevant in terms of I/O or post-processing and visualization, and it is of ever-increasing importance since more and more HPC applications are on the data side.

To ensure the impact of this holistic idea, it was clear that having a set of projects in our Priority Program where some address this issue and others that one, and where they may collaborate or not, would not suffice. Therefore, SPPEXA's concept was to have a set of larger projects, or project consortia (research units—Forschergruppen), that would all have to address at least two of the six big topics with their research agenda; and that would all have to combine a relevant large-scale application with HPC-methodical advancements. This means that neither a merely domain-driven research ("improve my code, and this is a contribution to HPC in itself"), as we see it frequently in domain-driven research initiatives (Collaborative Research Centers in physics, life sciences, or engineering, e.g.), nor a generic purely algorithmic research ("if I improve my solver, this will help everyone"), as we see it frequently in mathematics- or informatics-driven research initiatives, would be allowed to find their place in SPPEXA. This was somewhat challenging, since we had to communicate this concept clearly and to convince potential applicants and reviewers that everyone should really comply with this agenda.

Furthermore, there is one property better known from Collaborative Research Centers than from Priority Programs: program-wide joint activities. For example, we wanted to have a vivid collaboration framework of cross-project workshops; networking with the big international programs; a focus on education also through fostering novel teaching formats or coding weeks and doctoral retreats for the doctoral candidates; gender-related activities to understand, evaluate and work towards a more gender-balanced research community; etc. This allowed for sharing mutual best practices in HPC for the mathematics- or informatics- or application-driven areas. Therefore, there was more coordination than we see in typical Priority Programs.

3 Funded Projects and Internal Structure

In the first funding phase, the following thirteen projects or project consortia were funded:[1]

CATWALK—A Quick Development Path for Performance Models. Felix Wolf (Darmstadt), Christian Bischof (Darmstadt), Torsten Hoefler (Zürich), Bernd Mohr (Jülich), and Gabriel Wittum (Frankfurt)
ESSEX—Equipping Sparse Solvers for Exa-scale. Gerhard Wellein (Erlangen), Achim Basermann (Köln), Holger Fehske (Greifswald), Georg Hager (Erlangen), and Bruno Lang (Wuppertal)
Exa-Dune—Flexible PDE Solvers, Numerical Methods, and Applications. Peter Bastian (Heidelberg), Olaf Ippisch (Clausthal), Mario Ohlberger (Münster), Christian Engwer (Münster), Stefan Turek (Dortmund), Dominik Göddeke (Stuttgart), and Oleg Iliev (Kaiserslautern)
ExaFSA—Exa-scale Simulation of Fluid-Structure-Acoustics Interactions. Miriam Mehl (Stuttgart), Hester Bijl (Delft), Sabine Roller (Siegen), Dörte Sternel (Darmstadt), and Thomas Ertl (Stuttgart)
EXAHD—An Exa-Scalable 2-Level Sparse Grid Approach for Higher-Dimensional Problems in Plasma Physics and Beyond. Dirk Pflüger (Stuttgart), Hans-Joachim Bungartz (München), Michael Griebel (Bonn), Markus Hegland (Canberra), Frank Jenko (Garching), and Hermann Lederer (Garching)
EXAMAG—Exa-scale Simulations of the Evolution of the Universe Including Magnetic Fields. Volker Springel (Heidelberg) and Christian Klingenberg (Würzburg)
ExaSolvers—Extreme-scale Solvers for Coupled Problems. Lars Grasedyck (Aachen), Wolfgang Hackbusch (Leipzig), Rolf Krause (Lugano), Michael Resch (Stuttgart), Volker Schulz (Trier), and Gabriel Wittum (Frankfurt)
EXASTEEL—Bridging Scales for Multiphase Steels. Daniel Balzani (Bochum), Axel Klawonn (Köln), Oliver Rheinbach (Freiberg), Jörg Schröder (Duisburg-Essen), and Gerhard Wellein (Erlangen)
ExaStencils—Advanced Stencil-Code Engineering. Christian Lengauer (Passau), Armin Größlinger (Passau), Ulrich Rüde (Erlangen), Harald Köstler (Erlangen), Sven Apel (Saarbrücken), Jürgen Teich (Erlangen), Frank Hannig (Erlangen), and Matthias Bolten (Wuppertal)
FFMK—A Fast and Fault-tolerant Microkernel-Based System for Exa-scale Computing. Hermann Härtig (Dresden), Alexander Reinefeld (Berlin), Amnon Barak (Jerusalem), and Wolfgang E. Nagel (Dresden)
GROMEX—Unified Long-range Electrostatics and Dynamic Protonation for Realistic Biomolecular Simulations on the Exa-scale. Helmut Grubmüller (Göttingen), Holger Dachsel (Jülich), and Berk Hess (Stockholm)
DASH—Smart Data Structures and Algorithms with Support for Hierarchical Locality. Karl Fürlinger (München), Colin W. Glass (Stuttgart), José Gracia (Stuttgart), and Andreas Knüpfer (Dresden)
Terra-Neo—Integrated Co-Design of an Exa-scale Earth Mantle Modeling Framework. Hans-Peter Bunge (München), Ulrich Rüde (Erlangen), Gerhard Wellein (Erlangen), and Barbara Wohlmuth (München)

[1] Some Principal Investigators have changed affiliation during the SPPEXA program. We specified the most recent main affiliation here.

www.dbooks.org

After 3 years, twelve of those got a prolongation for the second funding phase, some with an "international extension" (bi-national with Japanese partners or tri-national with French and Japanese partners):

ESSEX-2—Equipping Sparse Solvers for Exa-scale. Gerhard Wellein (Erlangen), Achim Basermann (Köln), Holger Fehske (Greifswald), Georg Hager (Erlangen), Bruno Lang (Wuppertal), Tetsuya Sakurai (Tsukuba; Japanese partner), and Kengo Nakajima (Tokyo; Japanese partner)
Exa-Dune—Flexible PDE Solvers, Numerical Methods, and Applications. Peter Bastian (Heidelberg), Olaf Ippisch (Clausthal), Mario Ohlberger (Münster), Christian Engwer (Münster), Stefan Turek (Dortmund), Dominik Göddeke (Stuttgart), and Oleg Iliev (Kaiserslautern)
ExaFSA—Exa-scale Simulation of Fluid-Structure-Acoustics Interactions. Miriam Mehl (Stuttgart), Alexander van Zuijlen (Delft), Thomas Ertl (Stuttgart), Sabine Roller (Siegen), Dörte Sternel (Darmstadt), and Hiroyuki Takizawa (Tohoku; Japanese partner)
EXAHD—An Exa-Scalable 2-Level Sparse Grid Approach for Higher-Dimensional Problems in Plasma Physics and Beyond. Dirk Pflüger (Stuttgart), Hans-Joachim Bungartz (München), Michael Griebel (Bonn), Markus Hegland (Canberra), Frank Jenko (Garching), and Tilman Dannert (Garching)
EXAMAG—Exa-scale Simulations of the Magnetic Universe. Volker Springel (Heidelberg), Christian Klingenberg (Würzburg), Naoki Yoshida (Tokyo; Japanese partner), and Philippe Helluy (Strasbourg; French partner)
ExaSolvers—Extreme-scale Solvers for Coupeld Problems. Lars Grasedyck (Aachen), Rolf Krause (Lugano), Michael Resch (Stuttgart), Volker Schulz (Trier), Gabriel Wittum (Frankfurt), Arne Nägel (Frankfurt), Hiroshi Kawai (Tokyo; Japanese partner), and Ryuji Shioya (Toyo; Japanese partner)
EXASTEEL-2—Dual Phase Steels—From Micro to Macro Properties. Daniel Balzani (Bochum), Axel Klawonn (Köln), Oliver Rheinbach (Freiberg), Jörg Schröder (Duisburg-Essen), Olaf Schenk (Lugano), and Gerhard Wellein (Erlangen)
ExaStencils—Advanced Stencil-Code Engineering. Christian Lengauer (Passau), Ulrich Rüde (Erlangen), Harald Köstler (Erlangen), Sven Apel (Saarbrücken), Jürgen Teich (Erlangen), Frank Hannig (Erlangen), Matthias Bolten (Wuppertal), and Shigeru Chiba (Tokyo; Japanese partner)
FFMK—A Fast and Fault-tolerant Microkernel-Based System for Exa-scale Computing. Hermann Härtig (Dresden), Alexander Reinefeld (Berlin), Amnon Barak (Jerusalem), and Wolfgang E. Nagel (Dresden)
GROMEX—Unified Long-range Electrostatics and Dynamic Protonation for Realistic Biomolecular Simulations on the Exa-scale. Helmut Grubmüller (Göttingen), Holger Dachsel (Jülich), and Berk Hess (Stockholm)
DASH—Smart Data Structures and Algorithms with Support for Hierarchical Locality. Karl Fürlinger (München), Colin W. Glass (Stuttgart), José Gracia (Stuttgart), and Andreas Knüpfer (Dresden)
Terra-Neo—Integrated Co-Design of an Exa-scale Earth Mantle Modeling Framework. Hans-Peter Bunge (München), Ulrich Rüde (Erlangen), and Barbara Wohlmuth (München)

Furthermore, four new project consortia joined SPPEXA:

ADA-FS—Advanced Data Placement via Ad-hoc File Systems at Extreme Scales. Wolfgang E. Nagel (Dresden), André Brinkmann (Mainz), and Achim Streit (Karlsruhe)
AIMES—Advanced Computation and I/O Methods for Earth-System Simulations. Thomas Ludwig (Hamburg), Thomas Dubos (Versailles; French partner), Naoya Maruyama (RIKEN; Japanese partner), and Takayuki Aoki (Tokyo; Japanese partner)
ExaDG—High-order Discontinuous Galerkin for the Exa-scale. Guido Kanschat (Heidelberg), Katharina Kormann (München), Martin Kronbichler (München), and Wolfgang A. Wall (München)
MYX—MUST Correctness Checking for YML and XMP Programs. Matthias S. Müller (Aachen), Serge Petiton (Lille; French partner), Nahid Emad (Versailles; French partner), Taisuke Boku (Tsukuba; Japanese partner), and Hitoshi Murai (RIKEN; Japanese partner)

Finally, 1 year later, a seventeenth project joined SPPEXA as associated project:

ExtraPeak—Automatic Performance Modeling of HPC Applications. Felix Wolf (Darmstadt) and Torsten Hoefler (Zürich)

Hence, overall, there have been four Japanese-German and three French-Japanese-German consortia within SPPEXA. On the German side, an overall sum of 57 principal investigators from 39 institutions have been involved, representing informatics (25), mathematics (19), engineering (8), natural sciences (4), and life sciences (1).

Concerning governance, SPPEXA was headed by its two Spokespersons Hans-Joachim Bungartz (Technical University of Munich—TUM) and Wolfgang E. Nagel (Technical University of Dresden). For the everyday organization, a Program Coordinator (in chronological order: Benjamin Peherstorfer, now professor at New York University; Philipp Neumann, now professor at Helmut-Schmidt-University Hamburg; Benjamin Uekermann, now with Eindhoven University of Technology; and Severin Reiz, TUM) as well as an Office were established (both at TUM). Strategic decisions in SPPEXA were taken by the Steering Committee, consisting of H.-J. Bungartz, W. E. Nagel, as well as Sabine Roller (Siegen), Christian Lengauer (Passau), Hans-Peter Bunge (München), Dörte Sternel (Darmstadt), and—in the second funding phase—Nahid Emad (France) and Takayuki Aoki (Japan). Finally, a Scientific Advisory Board supported our activities and planning: George Biros (University of Texas at Austin), Rupak Biswas (NASA), Klaus Becker (Airbus), Rob Schreiber (at that time HP Labs), and Craig Stewart (University of Indiana at Bloomington).

www.dbooks.org

4 SPPEXA Goes International

Extreme-scale HPC has always been an international endeavor. In 2010, as the first call in the framework of the G8 Research Councils' Initiative on Multilateral Research Funding, the topic Application Software towards Exa-scale Computing for Global Scale Issues had been selected. In the sequel of that initiative, the idea arose to give SPPEXA in its second funding phase a more international flavor, beyond the individual international partners present in some of the consortia. DFG's head office contacted several of their partner institutions in other countries. While it turned out to be complicated to synchronize activities with the National Science Foundation (NSF) in the U.S., the discussions with the French Agence Nationale de la Recherche (ANR) and the Japan Science and Technology Agency (JST) became very concrete. Finally, for the first time, a funding phase of a complete DFG Priority Program was linked to funding formats from two other countries, and the three agencies combined their forces in a joint call run by DFG. Due to formal restrictions, two new types of SPPEXA consortia were open for application: bi-national Japanese-German or tri-national French-Japanese-German ones.

Overall, the following French institutions participated in SPPEXA projects: Université de Versailles, Université de Strasbourg, and Maison de la Simulation, Saclay. From the Japanese side, the involved partner institutions involved were RIKEN, Tokyo University of Technology, University of Tsukuba, University of Tokyo, Tohoku University, Tokyo University of Science, and Toyo University. Beyond research in the single consortia, one SPPEXA doctoral retreat was held in France, and SPPEXA co-organized three French-Japanese-German workshops—the first one 2017 in the French embassy in Tokyo, the second one in 2018 in the German embassy in Tokyo, and the third one in 2019, again in the French embassy. The first two focused on exa-scale computing, while the third one did a move towards artificial intelligence (AI) and, in particular, addressed the convergence of AI and HPC.

Further internationalization measures were the SPPEXA guest program, the research stays for doctoral candidates (up to 3 months; overall 25 taken in funding phase 2), and our PR activities at the big international meetings. For example, SPPEXA organized panels or sessions at the Supercomputing Conference (SC) and the International Supercomputing Conference (ISC HPC) and participated in the session and poster exhibition on DFG-funded collaborative research at DATE 2019.

5 Joint Coordinated Activities

As mentioned above, SPPEXA featured a rich program of joint cross-consortium activities (the following numbers refer to funding phase 2, 2016–2019):

Guests Overall, more than 85 guest researchers visited one or more SPPEXA projects.

Workshops Workshops were a particular format to foster exchange and collaboration across project consortia. Central funds had been established for that, and each SPPEXA PI could hand in proposals (two calls per year). The proposal had to depict how the cross-consortium effect was to be ensured (more than one organizing consortium, etc.). Overall, 41 SPPEXA workshops, held at conferences or standalone, were supported via this channel.

Doctoral Retreats The SPPEXA Doctoral Retreat had two main goals—first, to offer an additional educational component to our doctoral candidates; second, to overcome the sometimes narrow borders of research by connecting with international researchers on a doctoral level (guest lectures, own contributions, hands-on sessions, ...). Overall, three doctoral retreats were organized: Strasbourg (2016), Dresden (2017), and Wuppertal (2018).

Doctoral Research Stays Following the successful model of TUM Graduate School, where each doctoral candidate university-wide can get funds for an international research stay of up to 3 months, we encouraged our doctoral candidates SPPEXA-wide to enrich their PhD phase with such an international component. Overall, 25 such research interns were funded, examples for destinations being ETH Zurich, NORCE Bergen, or University of Tennessee.

Gender Activities Looking at the gender situation in HPC, it is obvious that the presence of women is even worse than in general in informatics. To improve that situation and to provide a more open atmosphere, a couple of measures were taken. At every Annual Plenary Meeting (2016, 2017, 2018, and 2019), we organized gender trainings by external coaches to raise awareness of gender biases in academia, each with 25 participants. Additionally, SPPEXA members organized workshop-like events such as student MINT mentoring days (2016–2018) and women's networking events in 2019. Moreover, we connected to industry (Bosch and IBM) via gender bias discussion days called "Equality at Exascale". Exceptional at this event was that not only women participated, but we had an ideal gender-parity in participants.

Impact on Education As a side effect, HPC education also got a boost by SPPEXA. Numerous lectures and lab courses were updated, and a lot of student theses had topics directly related to SPPEXA projects.

Prizes During the second phase of SPPEXA, every year, the best student and doctoral theses SPPEXA-wide were awarded a prize. Over the years, the winners were:

2016: Klaudius Scheufele (Stuttgart, master's thesis) and Benjamin Uekermann (Munich, PhD thesis);
2017: Sebastian Schweikl (Passau, bachelor's thesis), Simon Schwitanski (Aachen, master's thesis), and Moritz Kreutzer (Erlangen, PhD thesis);
2018/2019: Piet Jarmatz (Munich/Hamburg, master's thesis) and Sebastian Kuckuk and Christian Schmitt (Erlangen, PhD thesis).

www.dbooks.org

Support of Young Researchers For sustainability in academia, supporting young aspiring researchers is indispensable. We took measures by funding research stays for doctoral candidates and awarding prizes for exceptional theses. Additionally, we also supported bachelor and master students for the student cluster competition at the (international) supercomputing conferences SC and ISC HPC 2016–2019.

Public Relations Dissemination of research becomes more and more important. Continuing efforts from the first phase, SPPEXA featured articles in the InSiDE magazine, published by the GAUSS Center for Supercomputing, twice per year in 2016, 2017, and 2018 introducing one project each time. Furthermore, starting 2018, SPPEXA contributed five articles to the online platform Science Node.[2] Last, in 2018, SPPEXA also featured an article in the EU Research magazine.

Internationalisation See previous Sect. 4.

6 HPC Goes Data

The computational revolution goes on! Computers and sophisticated computational methods have shaped the "third paradigm", the third path to insight in science, complementing the classical approaches, theory and experiment, but also building a bridge and providing the missing link between those two. An early incarnation of "computational" were numerical simulations, later expanded by so-called "outer-loop scenarios", in which repeated simulations allow for enhanced results: optimization, parameter identification, stochastics, or uncertainty quantification. All of this, basically, was model-driven, following a deductive regime of model hypotheses and derivations from them. The latest appearance of "computational" can be characterized by the focus on data: data-enhanced simulation, data analytics, machine learning, or artificial intelligence. Instead of being based on models, this approach is much more data-driven, following an inductive regime of collecting data and drawing conclusions from them. In simplified words, the "data from models" turned into, or was complemented by, a "models from data". Despite that shift of focus, the basic underlying principle did not change: state-of-the-art computer systems and state-of-the-art computational methods are combined and used to advance the frontier of science. Something new is maybe the fact that the club of scientific domains that benefit from the "third paradigm" has become bigger: While numerical simulation was, more or less, driven by natural, engineering, and life sciences, the data-centered approach comprises all domains, including social sciences and humanities.

Of course, this development has a huge impact on HPC. In particular, new fields and new types of applications popped up, as well as new lines of architectures and systems. For example, in 2018, the majority of finalists for the Gordon Bell

[2] https://sciencenode.org/feature/the-race-to-exascale.php.

Award, the most renowned prize in HPC, already had a significant amount of machine learning in their papers. World-wide, HPC centers observe an increasing share of data-driven jobs on their machines. This is not surprising: as science and science methodology evolve, the kind of studies done in that context also does. Despite all those changes, the role of HPC is astonishingly stable: HPC is a core enabling technology of "computational". It was and still is an enabler of numerical simulation, and it has become a crucial enabler of data analytics and artificial intelligence. If artificial intelligence, machine learning, or deep learning have become so popular recently, this is much more due to the fact that established methodology can succeed due to HPC, than due to new AI/ML/DL methodology itself.

These developments are also visible at the end of SPPEXA. Several consortia already are on that "data-driven track", as, for example, our third French-Japanese-German workshop in Tokyo showed.

7 Shaping the Landscape

When SPPEXA started in 2013, the core idea was to significantly improve algorithms, software, and tools, in order to be prepared for the exa-scale age. In the meantime, we are at the eve of exa-scale systems, as the co-design developments in the U.S. and in Japan (Fugaku) or the discussions in the European Union on exa-scale and pre-exa-scale systems show. And research in SPPEXA has definitely contributed to the application landscape in Germany being much closer to "exa-scale-readiness" than before. Several leading application software packages were involved, and significant progress in terms of scalability and parallel efficiency could be achieved. Furthermore, and maybe even more important, the SPPEXA consortia showed the advantages of the multi-disciplinary engagement, brought together a lot of groups and ideas disconnected before, and, thus, justified the concept of larger, cross-institutional, and cross-disciplinary teams instead of single-PI projects.

The visibility SPPEXA got is stunning. SPPEXA was present at the leading international conferences (Euro-Par, Supercomputing, ISC HPC)—through individual presentations and special events, such as minisymposia or panels. But also at "neighboring" events, such as the DATE 2019 (Design, Automation, and Test in Europe), SPPEXA had a presentation slot and a booth. SPPEXA was involved in the activities (workshops, white papers, etc.) of the BDEC Community (Big Data and Extreme-Scale Computing) as well as in the organization of the Long Program "Science at Extreme Scales: Where Big Data Meets Large-scale Computing" at the Institute for Pure and Applied Mathematics (IPAM) in Los Angeles, and it co-organized a French-Japanese-German workshop series in Tokyo (cf. the section on internationalization). Thus, at an international scale, SPPEXA was generally perceived as the "German player" in the HPC software concert.

www.dbooks.org

8 Concluding Remarks

Without any doubt, SPPEXA has written a success story: in terms of its research, concerning the innovative funding format, with its multi-disciplinary approach, its multi-national facets, and—last, but not least—its huge visibility. We are grateful for all the support we got from the German Research Foundation (DFG): the funding, but also for the encouragement during the preparation of SPPEXA and the continued advice during its runtime.

Appendix 1: Qualification

The following achievements have been completed in the SPPEXA program within 1.1.2016 and 30.04.2020:

Projects	Completed PhD theses	Completed habilitations	Calls to professorship
AIMES	0	0	1
ADA-FS	0	0	0
DASH	1	0	1
ESSEX	1	1	0
ExaDG	4	0	1
Exa-Dune	4	0	1
ExaFSA	2	0	0
EXAHD	3	0	0
EXAMAG	9	0	0
ExaSolvers	1	0	1
EXASTEEL	2	0	1
ExaStencils	5	2	4
ExtraPeak	3	0	0
FFMK	1	0	0
GROMEX	2	0	0
MYX	0	0	0
Terra-Neo	3	0	0
Coordination	1	1	2
Overall	43	3	12

The previous table follows the DFG requirements for final reports in priority programs. At least 25 additional PhD candidates are close to being finished; however, due to the lengthy defense procedure they are not counted here.

Also, please take into account that project consortia vary in size (regarding Principal Investigators and PhD candidates) and their start/end date.

Appendix 2: Software from Project Consortia

In the following, a table with links to software that has been developed by the project consortia in SPPEXA Phase-II is given.

Project	Software developed
AIMES	SCIL
	github.com/JulianKunkel/scil
ADA-FS	GekkoFS
	tu-dresden.de/zih/forschung/projekte/ada-fs
DASH	DASH
	www.dash-project.org/
ESSEX	PHIST, GHOST, CRAFT, RACE, ScaMaC
	bitbucket.org/essex/{PHIST, ..., RACE, matrixcollection}
ExaDG	deal.II
	github.com/dealii/dealii
EXA-Dune	DUNE
	gitlab.dune-project.org/exadune
ExaFSA	preCICE
	github.com/precice/precice
	Ateles
	apes.osdn.io/pages/ateles
EXAHD	SG++
	github.com/SGpp/SGpp
EXAMAG	AREPO
	arepo-code.org/
ExaSolvers	utopia
	bitbucket.org/zulianp/utopia/src/master/
EXASTEEL	FE2TI
	www.numerik.uni-koeln.de/14079.html
ExaStencils	LFA Lab
	hrittich.github.io/lfa-lab/
	ExaSlang
	i10git.cs.fau.de/exastencils/release
ExtraPeak	Extra-P
	www.scalasca.org/scalasca/software/extra-p/
FFMK	FFMK
	ffmk.tudos.org/
GROMEX	GROMACS
	www.gromacs.org

www.dbooks.org

MYX	MUST
	doc.itc.rwth-aachen.de/display/CCP/Project+MUST
Terra-Neo	HyTeG
	i10git.cs.fau.de/hyteg/hyteg
	waLBerla
	www.walberla.net/
	TerraNeo
	terraneo.fau.de/

Appendix 3: Project Consortia Key Publications

This volume represents a continuation of the corresponding report in SPPEXA Phase-I, which is referenced several times in the text above:

1. Bungartz, H.-J., Neumann, P., Nagel, W.E.: *Software for Exascale Computing-SPPEXA 2013–2015*, vol. 113. Springer, Berlin (2016)

SPPEXA Phase-II showed visibility in the research community with numerous publications. In the following we provide a list of two key publications for each project consortium:[3]

AIMES

1. Jum'ah, N., Kunkel, J.: Performance portability of earth system models with user-controlled GGDML code translation. In: International Conference on High Performance Computing, pp. 693–710. Springer, Berlin (2018)
2. Kunkel, J., Novikova, A., Betke, E., Schaare, A.: Toward decoupling the selection of compression algorithms from quality constraints. In: International Conference on High Performance Computing, pp. 3–14. Springer, Berlin (2017)

ADA-FS

1. Vef, M.A., Moti, N., Süß, T., Tocci, T., Nou, R., Miranda, A., Cortes, T., Brinkmann, A.: GekkoFS—a temporary distributed file system for HPC applications. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 319–324. IEEE, Piscataway (2018)
2. Soysal, M., Berghoff, M., Klusáček, D., Streit, A.: On the quality of wall time estimates for resource allocation prediction. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops, pp. 1–8. ACM, New York (2019)

DASH

1. Kowalewski, R., Jungblut, P., Fürlinger, K.: Engineering a distributed histogram sort. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER), pp. 1–11. IEEE, Piscataway (2019)

[3]Following the DFG requirements for final reports in priority programs.

2. Fürlinger, K., Glass, C., Gracia, J., Knüpfer, A., Tao, J., HHünichnich, D., Idrees, K., Maiterth, M., Mhedheb, Y., Zhou, H.: DASH: data structures and algorithms with support for hierarchical locality. In: European Conference on Parallel Processing, pp. 542–552. Springer, Berlin (2014)

ESSEX

1. Pieper, A., Kreutzer, M., Alvermann, A., Galgon, M., Fehske, H., Hager, G., Lang, B., Wellein, G.: High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. J. Comput. Phys. **325**, 226–243 (2016)
2. Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H.: Increasing the performance of the Jacobi–Davidson method by blocking. SIAM J. Sci. Comput. **37**(6), C697–C722 (2015)

ExaDG

1. Kronbichler, M., Kormann, K.: Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM Trans. Math. Softw. **45**(3), 1–40 (2019)
2. Fehn, N., Wall, W.A., Kronbichler, M.: Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. Int. J. Numer. Methods Fluids **88**(1), 32–54 (2018)

EXA-Dune

1. Bastian, P., Engwer, C., Göddeke, D., Iliev, O., Ippisch, O., Ohlberger, M., Turek, S., Fahlke, J., Kaulmann, S., Steffen Müthing, S., et al.: EXA-DUNE: flexible PDE solvers, numerical methods and applications. In: European Conference on Parallel Processing, pp. 530–541. Springer, Berlin (2014)
2. Engwer, C., Altenbernd, M., Dreier, N.A., Göddeke, D.: A high-level C++ approach to manage local errors, asynchrony and faults in an MPI application. In: 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 714–721. IEEE, Piscataway (2018)

ExaFSA

1. Mehl, M., Uekermann, B., Bijl, H., Blom, D., Gatzhammer, B., Van Zuijlen, A.: Parallel coupling numerics for partitioned fluid–structure interaction simulations. Comput. Math. Appl. **71**(4), 869–891 (2016)
2. Totounferoush, A., Pour, N.E., Schröder, J., Roller, S., Mehl, M.: A new load balancing approach for coupled multi-physics simulations. In: 2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 676–682. IEEE, Piscataway (2019)

EXAHD

1. Obersteiner, M., Hinojosa, A.P., Heene, M., Bungartz, H.J., Pflüger, D.: A highly scalable, algorithm-based fault-tolerant solver for gyrokinetic plasma simulations. In: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, pp. 1–8 (2017)

www.dbooks.org

2. Hupp, P., Heene, M., Jacob, R., Pflüger, D.: Global communication schemes for the numerical solution of high-dimensional PDEs. Parallel Comput. **52**, 78–105 (2016)

ExaSolvers

1. Benedusi, P., Garoni, C., Krause, R., Li, X., Serra-Capizzano, S.: Space-time FE-DG Discretization of the anisotropic diffusion equation in any dimension: the spectral symbol. SIAM J. Matrix Anal. Appl. **39**(3), 1383–1420 (2018)
2. Kreienbuehl, A., Benedusi, P., Ruprecht, D., Krause, R.: Time-parallel gravitational collapse simulation. Commun. Appl. Math. Comput. Sci. **12**(1), 109–128 (2015)

ExaStencils

1. Köstler, H., Schmitt, C., Kuckuk, S., Kronawitter, S., Hannig, F., Teich, J., Rüde, U., Lengauer, C.: A scala prototype to generate multigrid solver implementations for different problems and target multi-core platforms. Int. J. Comput. Sci. Eng. **14**(2), 150–163 (2017). https://doi.org/10.1504/IJCSE.2017.082879
2. Schmitt, C., Kronawitter, S., Hannig, F., Teich, J., Lengauer, C.: Automating the development of high-performance multigrid solvers. Proc. IEEE **106**(11), 1969–1984 (2018)

ExtraPeak

1. Shudler, S., Calotoiu, A., Hoefler, T., Wolf, F.: Isoefficiency in practice: configuring and understanding the performance of task-based applications. In: ACM SIGPLAN Notices, vol. 52, pp. 131–143. ACM, New York (2017)
2. Calotoiu, A., Hoefler, T., Poke, M., Wolf, F.: Using automated performance modeling to find scalability bugs in complex codes. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 45. IEEE, Piscataway (2013)

FFMK

1. Weinhold, C., Lackorzynski, A., Härtig, H.: FFMK: an HPC OS based on the L4Re microkernel. In: Operating Systems for Supercomputers and High Performance Computing, pp. 335–357. Springer, Berlin (2019)
2. Gholami, M., Schintke, F.: Multilevel checkpoint/restart for large computational jobs on distributed computing resources. In: IEEE 38th Symposium on Reliable Distributed System (SRDS) (2019)

GROMEX

1. Beckmann, A., Kabadshow, I.: Portable node-level performance optimization for the fast multipole method. In: Recent Trends in Computational Engineering-CE2014, pp. 29–46. Springer, Berlin (2015)
2. Kutzner, C., Páll, S., Fechner, M., Esztermann, A., de Groot, B.L., Grubmüller, H.: More bang for your buck: Improved use of GPU nodes for GROMACS 2018. J. comput. chem. **40**(27), 2418–2431 (2019)

MYX

1. Protze, J., Tsuji, M., Terboven, C., Dufaud, T., Murai, H., Petiton, S., Emad, N., Müller, M., Boku, T.: Myx—runtime correctness analysis for multi-level parallel programming paradigms. In: Software for Exascale Computing: SPPEXA 2016–2019. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2020)
2. Protze, J., Schulz, M., Ahn, D.H., Müller, M.S.: Thread-local concurrency: a technique to handle data race detection at programming model abstraction. In: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, pp. 144–155 (2018)

Terra-Neo

1. Bauer, S., Huber, M., Ghelichkhan, S., Mohr, M., Rüde, U., Wohlmuth, B.: Large-scale simulation of mantle convection based on a new matrix-free approach. J. Comput. Sci. **31**, 60–76 (2019)
2. Huber, M., Gmeiner, B., Rüde, U., Wohlmuth, B.: Resilience for massively parallel multigrid solvers. SIAM J. Sci. Comput. **38**(5), S217–S239 (2016)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

A Perspective on the SPPEXA Collaboration from France



Nahid Emad

As the French member of the Steering Committee of SPPEXA, it is my great pleasure to give a short address to this volume from the perspective of the French partners in this German-French-Japanese cooperation. To highlight the types of software supported by SPPEXA, we first present a classification of high-performance software types. We then take a look at the recent activities of HPC software in France under the SPPEXA umbrella. Next, some local impacts of the SPPEXA collaboration on the French HPC community is provided, and lastly, an outlook to future collaborations.

1 HPC Software in Three Phases

High-performance numerical software targets at obtaining relevant scalability in space and time for large-size applications by using a large number of cores/processors/nodes of powerful computers. They can be classified into three phases: pre-treatment, treatment, and post-treatment. Obviously, such a software often belongs to more than one of the categories mentioned.

Pre-processing Software The precise definition of these phases depends on the context, but the main role of pre-treatment software is the preparation of the input data for the treatment phase. This preparation sometimes consists of a rather complex parallel algorithmic and programming processing. Big data compression, uniform data formatting, and conditioning improvement of data matrices are some examples of the pre-treatment phase.

N. Emad (✉)
University of Paris Saclay, UVSQ, LI PARAD, Maison de la Simulation, Versailles, France
e-mail: nahid.emad@uvsq.fr

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_2

Treatment Software The HPC software in the processing phase concerns mainly high-performance numerical simulation of physical phenomena, social networks, etc. These softwares could be classified as (1) "standard" libraries and (2) ad-hoc libraries done by application field programmers, which implement their application by making use of building blocks (partially or totally) of the libraries of class (1). The latter implement numerical methods with the main parallel/distributed programming methodologies, such as ScaLAPACK, PETSc, SLEPc, ATLAS, etc. In the fields of application , HPC ad-hoc software targets epidemiology, electromagnetism, gamma astronomy, safety, or health and nutrition. Some ad-hoc software examples are CEDRE which targets simulation for energy and propulsion, CELESTIA and STELLARIUM which is a space simulator and a planetarium for observing the solar system and the rest of the universe in real time and in 3D, GAUSS, which is a flexible platform for data analysis, and AREPO, which is a cosmological hydrodynamical simulation code on a dynamic unstructured mesh.

Post-processing Software Post-processing software essentially involves the analysis, visualization, and performance evaluation of the treatment phase results. Some examples of such software are ParaView (a multi-platform data analysis and visualization application), VisIt (an interactive platform for visualization, animation and data analysis), MAQAO (sets of software tools for code optimization in the core or node level of a parallel architecture), and Maya (a software for modeling, simulation, and 3D animation).

All these software packages generally translate a physical phenomenon, social behaviour, etc. into mathematical equations. Their high-performance implementation on parallel and/or distributed systems is a delicate task and requires a huge ecosystem with people having interdisciplinary skills. This makes the existence and use of accompanying software necessary, which provides the logistics of high-performance computing. These software frameworks provide the environment for high-performance programming and often conceal the complexity of underlying parallel and/or distributed architectures. As a consequence this allows the users to focus on main objectives. MPI, OpenMP, Globus, Condor, XMP, YML, MUST, etc. are very few examples of this kind of software.

2 Trilateral Projects in SPPEXA and Their Impact

SPPEXA targeted fundamental research on different aspects of HPC software and covered software categories cited before with a co-design approach. Thanks to ANR, DFG, and JST, the trilateral French/German/Japanese projects have been funded within SPPEXA. Some of these projects, such as MYX, have benefited from pre-existing bilateral collaborations. This allowed a dynamical and productive work from the beginning and for a rapid progress towards the objectives set. In addition to project meetings, the cross-project SPPEXA workshops have given a new dynamic to the trilateral collaborations paving the way for the organization of other conferences, workshops or seminars.

www.dbooks.org

The EXAMAG (Exascale Simulations of the Magnetic Universe) project is an example of an SPPEXA trilateral French/German/Japanese project with the aim of improving the astrophysical moving-mesh code AREPO and extending its range of applicability for high scale computing platforms. EXAMAG is an ad-hoc HPC software of the processing category of the classification given in the previous section.

MYX (MUST Correctness Checking for YML and XMP Programs) also is a trilateral French/German/Japanese project which aims to offer a guideline how to limit the risk to introduce errors and how to best express the parallelism to catch errors at runtime. From a practical viewpoint, MYX aims at the design and the application of a scalable correctness checking tool MUST to YML and XMP. In the MYX project, the main developed software packages (YML, XMP and MUST) belong to the last category of HPC software; the ones providing the logistics of high-performance application programs. However, in order to validate the design and development of these softwares, many other benchmarks and/or real applications are developed. Among them are the multiple restarted Krylov methods/HPCS ad-hoc, matrix generator/pretreatment , epidemic HPCS ad-hoc, etc.

The SPPEXA funding of workshops with several projects involved added an extra dimension of interdisciplinarity. In collaboration with the DASH and ESSEX-II projects, MYX members organized four trilateral (German/Japanese/French) workshops. Two of them have been hold at university of Paris Saclay/Versailles in France. With the prominent invited speakers and the talks of SPPEXA-involved project members, these workshops have been very attractive (40 and 60 attendees, respectively). An important number of indirect outcomes of SPPEXA activities (workshops, "open" trilateral meetings, doctoral retreats, etc.) generated new connections between German and French colleagues and students. A few examples are ▷ the review of the PhD dissertation of a non-SPPEXA funded French student by Sabine Roller, professor at Siegen University, Germany, ▷ Xinzhe Wu, who finished recently his PhD, funded by ANR part of ANR/DFG/JST MYX project and, who is currently in a post-doctoral position at *Jülich Research Centre* in Germany, ▷ M.A. Diop, currently PhD student, funded by a French CIFRE followship (with ATOS/EVIDIAN company), who participated in SPPEXA doctoral retreats as well as several SPPEXA workshops, ▷ a workhop organised by Sabine Roller and Nahid Emad at HPC Asia, or a large number of BSc and MSc students benefiting from the collaboration.

3 What Will Be Next?

The on-going convergence between machine learning, data analysis, and high-performance computing is creating new algorithmic and co-design approaches that need to be taken into account for the future. With the three tri-national workshops in Tokyo, SPPEXA has contributed to this on-going development, and we are all looking forward to a continued collaboration in the future.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

A Perspective on the SPPEXA Collaboration from Japan



Takayuki Aoki

A national research project was running in Japan from 2010 to 2017 named "Development of System Software Technologies for Post-Peta Scheme High Performance Computing" (so called Post-Peta CREST) . It was supported by JST (Japan Science and Technology Agency), which is the Japanese counterpart to DFG ("Deutsche Forschungsgemeinschaft"). The Post-Peta CREST project was similar to the first funding phase of SPPEXA in the sense that it had a primarily national scope. Then, the Post-Peta CREST project opened up to international collaboration, and some projects were extended for two more years, where they formed collaborative research groups with SPPEXA phase-II projects. Projects with contributions from Japan are *ExaFSA*, *ExaStencils*, *EXAMAG*, *ESSEX-II*, *EXASOLVERS*, *AIMES*, and *MYX*, with more than 10 researchers in the second phase of SPPEXA. To highlight the success of the Japanese collaboration with SPPEXA, we have a brief look at two working groups.

ppOpen-HPC and ESSEX-II

As a part of Post-Peta CREST projects between 2011 and 2015, a group at the University of Tokyo developed ppOpen-HPC, which is an open source infrastructure for the development and execution of optimized and reliable simulation code on post-peta-scale (pp) parallel computers based on many-core architectures. The framework covers various types of procedures for scientific computations in various types of computational models, such as FEM, FDM, FVM, BEM, and DEM.

T. Aoki (✉)
Tokyo Institute of Technology, Tokyo, Japan
e-mail: taoki@gsic.titech.ac.jp

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_3

Automatic tuning (AT) technology enables automatic generation of optimized libraries and applications under various types of environments. The most updated version of ppOpen-HPC was released as open source software, which is available at https://github.com/Post-Peta-Crest/ppOpenHPC.

In 2016, the team of ppOpen-HPC joined the SPPEXA phase-II project ESSEX-II including members from the University of Erlangen-Nuremberg, which is funded by JST-CREST and SPPEXA under Japan (JST)-Germany (DFG) collaboration until 2018. ESSEX-II developed pK-Open-HPC (extended version of ppOpen-HPC, a framework for exa-feasible applications), such as preconditioned iterative solvers for quantum science.

Sparse coefficient matrices derived from applications in quantum science have generally relatively very small diagonal components, and they are generally ill-conditioned. Therefore, it is difficult to apply preconditioned iterative methods developed in ppOpen-HPC directly to such applications. The ESSEX-II team developed a regularization method for robustness based on blocking and diagonal shifting, which provide efficient and robust convergence of ill-conditioned problems in quantum science. Preconditioning methods with the regularization method are implemented in GHOST/PHIST libraries for solving matrices, which integrates all linear solvers and related methods developed in ESSEX/ESSEX-II projects. Moreover, they proposed a new method for global parallel reordering, which provides robust and efficient convergence of parallel iterative solvers with ILU-based preconditioning for very ill-conditioned problems. The developed method kept iteration number constant in strong scaling cases up to O(104) MPI processes for very ill-conditioned problems. This is the first method for global parallel reordering.

In the ESSEX-II project, CRAFT (A library for application-level Checkpoint/Restart and Automatic Fault Tolerance) has been developed for fault resilience on exascale systems by checkpointing. ESSEX-II integrated the dynamic load balancing function and CRAFT, and developed a prototype of a fault-resilient framework for parallel FEM applications. Parallel FEM codes can continue computations by this framework, when some of the computing nodes fail. This framework does not need spare nodes for fault resilience. This idea can be extended to various types of procedures for dynamic scheduling on exascale systems.

Collaborations in ESSEX-II project have been continuing in the JHPCN projects ("Numerical Library with High-Performance/Adaptive-Precision/High-Reliability" (starting in 2018), "Innovative Multigrid Methods" (starting in 2018)), and in "Innovative Methods for Scientific Computing in the Exascale Era by Integrations of (Simulation+Data+Learning)" funded by "Grant-in-Aid for Scientific Research (S) (KAKENHI S)" (2019-2023)

www.dbooks.org

Xevolver and ExaFSA

The so-called Xevolver project is one of the Post-Peta CREST projects from 2011 to 2017. A group at the Tohoku University discussed how they could help in legacy code migration to future-generation extreme-scale computing systems that will be massively parallel and heterogeneous. Even today an HPC application code is likely optimized assuming a particular system configuration, and hence specialized only for its target system. In general, such an application is not performance-portable at all. As the HPC system architectures are now diverging and also getting more complicated in terms of accelerators, it will require more time and effort to migrate or re-optimize the code to another system in the future. To make matters worse, system-specific code optimizations are tightly interwoven with the computation and thereby degrade the code readability and maintainability, even though HPC applications need to evolve not only for achieving high performance, but also for advancing computational science. Therefore, in the project, our team has developed a code transformation framework, Xevolver, so that users can define their own code transformations and thus express system-specific code optimizations as code transformation rules. Since code transformation rules can be defined separately from application codes themselves, the Xevolver framework can contribute to separation of system-specific performance concerns from application codes, and hence prevent overcomplicating the codes.

In 2016, core members of the Xevolver research team joined the second phase of the ExaFSA project in order to demonstrate that the Xevolver approach is effective for optimizing real-world applications in practice. The Xevolver approach assumes that an HPC application is developed by a team work of at least two kinds of programmers. One is application developers and the other is performance engineers. Application developers are interested in simulation results rather than performance, while performance engineers are mainly focusing on sustained simulation performance. Therefore, Japanese researchers have worked as performance engineers using Xevolver by considering German research groups as application developers.

The ExaFSA project focused on engineering two solvers, FASTEST and Ateles, which have been developed in the ExaFSA project as primary building blocks of a practical coupled simulation. An incompressible flow solver, FASTEST, has a long history of development and was once optimized for classic vector machines. Thus, some of important kernels still have two versions, default version and its vector-optimized version. In the ExaFSA project, hence, they used the Xevolver framework to express the differences between the two versions, and demonstrated that the vector-optimized version can be generated by transforming the default version. That is, the Xevolver approach can express the system-specific code optimizations as code transformation rules, and thus even simplify the code while achieving high performance and portability. Ateles is based on based on Discontinuous Galerkin (DG) discretization method, and a part of the simulation framework, APES, was developed at the University of Siegen in Germany. Unlike FASTEST, Ateles is written using modern Fortran language features to hide the implementation

details. However, the kernel loops still need to be optimized in different ways for individual system architectures to achieve high performance. For example, some loop optimizations with compiler directives are mandatory for the NEC SX-ACE vector computing system to properly vectorize and thus efficiently execute the loops. In this project, Xevolver is used to apply the loop optimizations without major modifications of the original code. Accordingly, the ExaFSA project was a very good opportunity for us to demonstrate that the Xevolver approach can help an appropriate division of labor between application developers and performance engineers by achieving separation of concerns. This clarification of role-sharing will be very helpful for long-term application development especially in an upcoming extreme-scale computing era.

The Role of Japan in HPC Collaborations

The SPPEXA program was unique in the sense that it established sustainable connections in the field of HPC between France, Germany, and Japan. With the supercomputing infrastructure in Japan (and its upcoming flagship supercomputer Fugaku), the three countries are suitable partners for portability and methodology comparisons and, thus, synergistic research developments (such as within SPPEXA connection).

A new field of interest in Japan, Germany, and France is data science and its connection to HPC. SPPEXA participated in the tri-lateral workshop in Tokyo "Convergence of HPC and Data Science for Future Extreme Scale Intelligent Applications", where we discussed new possible collaborations in the fields of HPC and Big Data. Looking back at SPPEXA, we see many success stories, and hope for a lot of continuing collaborations.

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

Part II
SPPEXA Project Consortia Reports

ADA-FS—Advanced Data Placement via Ad hoc File Systems at Extreme Scales



Sebastian Oeste, Marc-André Vef, Mehmet Soysal, Wolfgang E. Nagel, André Brinkmann, and Achim Streit

Abstract Today's High-Performance Computing (HPC) environments increasingly have to manage relatively new access patterns (e.g., large numbers of metadata operations) which general-purpose parallel file systems (PFS) were not optimized for. Burst-buffer file systems aim to solve that challenge by spanning an ad hoc file system across node-local flash storage at compute nodes to relief the PFS from such access patterns. However, existing burst-buffer file systems still support many of the traditional file system features, which are often not required in HPC applications, at the cost of file system performance.

The ADA-FS project aims to solve that challenge by providing a temporary burst-buffer file system—GekkoFS—which relaxes POSIX, based on previous usage studies of how HPC applications use file systems. Due to a highly distributed and decentralized design GekkoFS reaches scalable data and metadata performance with tens of millions of metadata operations per second on a 512 node cluster. The ADA-FS project further investigated the benefits of using ad hoc file systems and how they can be integrated into the workflow of supercomputing environments. In addition, we explored how to gather application-specific information to optimize the file system for an individual application.

S. Oeste (✉) · W. E. Nagel
TU Dresden, Dresden, Germany
e-mail: sebastian.oeste@tu-dresden.de; wolfgang.nagel@tu-dresden.de

M.-A. Vef · A. Brinkmann
Johannes Gutenberg University Mainz, Mainz, Germany
e-mail: vef@uni-mainz.de; brinkman@uni-mainz.de

M. Soysal · A. Streit
KIT Karlsruhe, Karlsruhe, Germany
e-mail: mehmet.soysal@kit.edu; achim.streit@kit.edu

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_4

www.dbooks.org

1 Introduction

Application-imposed workloads on *High-Performance computing* (HPC) environments have considerably changed in the past decade. While traditional HPC applications have been compute-bound, large-scale simulations, today's HPC applications are also generating, processing, and analyzing massive amounts of experimental data—known as *data-driven science* applications—affecting several scientific fields. Some of which have already made significant progress in previously unaddressable challenges due to newly discovered techniques [27, 55].

Many data-driven workloads are based on new algorithms and data structures which impose new requirements on HPC file systems [45, 77]. Particularly, large numbers of metadata operations, data synchronization, non-contiguous and random access patterns, and small I/O requests [14, 45], used in data-driven science applications, are challenging for today's general-parallel file systems (PFSs) to handle since past workloads mostly perform sequential I/O operations on large files. Not only are such applications disruptive to the shared storage system but also heavily interfere with other applications which access the same shared storage system [18, 68]. As a result, many workloads which impose these new types of I/O operations suffer from prolonged I/O latencies, reduced file system performance, and occasional long wait times.

Software-based approaches, e.g., application modifications or middleware and high-level libraries [21, 39], and hardware-based approaches, moving from magnetic disks to NAND-based solid-state drives (SSDs) within PFSs, are attempts to mitigate the impact of these new access patterns on the HPC system. However, software-based approaches often suffer from time-consuming adaptations within applications and are sometimes (based on the underlying algorithms) even impossible to adapt to. One of the hardware-based approaches leverages on, nowadays, existing SSDs, installed within a compute node, in order to use them as *node-local* burst buffers. To achieve high metadata performance, they can be deployed in combination with a dynamic *burst buffer file system* [5, 78]. Nonetheless, existing burst buffer file systems have been mostly POSIX compliant which can severely reduce a file system's peak performance [75].

The ADA-FS project, funded by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing", aims to further explore the possibilities of burst buffer file systems in this context while investigating how they can be used in a modern HPC system. The developed burst buffer file system—*GekkoFS*—acts as ADA-FS' main component. GekkoFS is a temporarily deployed, highly-scalable distributed file system for HPC applications which aims to accelerate I/O operations of common HPC workloads that are challenging for modern PFSs. As such, it can be used in several temporary use cases, such as the lifetime of a compute job or in longer-term use cases, e.g., campaigns. Unlike previous works on burst buffer file systems, it relaxes POSIX by removing some of the semantics that most impair I/O performance in a distributed context and takes previous studies on the behavior of HPC applications into account [37] to

optimize the most used file system operations. As a result, GekkoFS reaches scalable data and metadata performance with tens of millions of metadata operations per second on a 512 node cluster while still providing strong consistency for file system operations that target a specific file or directory. In fact, due to its highly distributed and decentralized file system design, GekkoFS is built to perform on even bigger supercomputers, as exascale environments are right around the corner.

While GekkoFS provides the core building block within ADA-FS, it relies and benefits from further information of the application it is used with. Application-specific information that we gather can then further optimize the file system (e.g., the used file system block size) and therefore may increase the file system's performance in terms of latency and throughput. In addition, the ADA-FS project investigated how such a temporary and *on demand* burst buffer file system can be integrated into the workflow of batch systems in supercomputing environments. Although it is hard to reliably predict when compute jobs finish to prematurely deploy GekkoFS for a following ADA-FS job, for instance, we investigated and showed the benefits of on demand burst buffer file systems concerning both application performance and the reduction of the PFS load as a result of using such a file system.

The article is structured as follows: first, we describe GekkoFS' design and its evaluation of nowadays common and challenging HPC workloads on a 512 node cluster in Sect. 2. Section 3 discusses the existing challenges when data is staged in advance and how we solved the challenge, through implementing a plugin for the batch system. Section 4 discusses how we can detect system resources like the amount of node local storage or the NUMA configuration of a node which can be used for the deployment of the GekkoFS file system even on heterogenous compute nodes. In Sect. 5 we show how the option for an on demand file system, can be added to an HPC system. We follow with an evaluation of the performance of GekkoFS for new NVME based storage systems in Sect. 6. Finally, we conclude in Sect. 7.

2 GekkoFS—A Temporary Burst Buffer File System for HPC

In this section, we present the main component of ADA-FS—GekkoFS. GekkoFS is a temporarily deployed, highly-scalable burst buffer file system for HPC applications. In general, the goal of GekkoFS is to accelerate I/O operations in common HPC workloads that are challenging for modern PFSs while offering the combined storage capabilities of node-local storage devices. Further, it does not only aim for providing scalable I/O performance, but, in particular, focuses on offering scalable metadata performance by departing from traditional ways of handling metadata in distributed file systems. To provide a single, global namespace, accessible to all file system nodes, the file system pools together fast node-local storage resources of all participating file system nodes.

www.dbooks.org

Based on previous studies [37] on the behavior of HPC applications, GekkoFS relaxes or removes some of the POSIX semantics, known to heavily impact I/O performance in a distributed environment. As a result, it is able to optimize for the most used file system operations, achieving tens of millions of metadata operations per second on a 512 node cluster. At the same time, GekkoFS is able to run complex applications, such as OpenFOAM solvers [32], and since the file system runs in user-space and it can be easily deployed in under 20 s on a 512 node cluster, it is usable by any user. Consequently, GekkoFS can be used for several use cases which require an ephemeral distributed file system, such as during the lifetime of a compute job or campaigns where data is simultaneously accessed by many nodes in short bursts.

Parts of this section's contents is build on the conference paper by the authors M.-A. Vef et al. [72] and the journal article by the authors M.-A. Vef et al. [71] which both discuss each of the system components of GekkoFS in more detail and provide an in-depth investigation into the performance of GekkoFS compared to other file systems in various HPC environments. First, Sect. 2.1 provides a background on parallel and distributed file systems and discusses some of the related work in the context of burst buffer file systems. Section 2.2 presents the file system's core architecture and design to achieve scalable data and metadata performance in a distributed environment. Finally, in Sect. 2.3 we demonstrate GekkoFS data and metadata performances.

2.1 Related Work

In this section, we give an overview over existing HPC file systems and discuss the differences to GekkoFS.

2.1.1 General-Purpose Parallel File Systems

Most HPC systems are equipped with a backend storage system which is globally accessible using a parallel file system (e.g., GPFS [57], Lustre [7, 53], BeeGFS [26], or PVFS [56]). These file systems offer a POSIX-like interface and focus on data consistency and long-term storage. However, due to the nature of the file system being globally accessible, single applications can disrupt the I/O performance of other applications as well. In addition, these file systems are not well suited for small file accesses, in particular on shared files, often found in scientific applications [45].

The design of GekkoFS does not focus on long-term storage and aims for temporary use cases, such as in the context of compute jobs or campaigns. In addition, since GekkoFS relaxes POSIX semantics, it is able to provide a significant increase in metadata performance.

2.1.2 Node-Local Burst Buffers

Burst buffers are fast, intermediate storage systems that aim to reduce the load on the global file system and on reducing an applications' I/O overhead [38]. Such burst buffers can be categorized into two groups [78]: remote-shared and node-local. Remote-shared burst buffers are generally dedicated I/O nodes to forward application I/O to the underlying PFS, e.g., DDN's IME[1] and Cray's DataWarp.[2]

Node-local burst buffers, on the other hand, are collocated with compute nodes, using existing node-local storage. This node-local storage is then used to create a (distributed) file system which spans over a number of nodes for the lifetime of a compute job, for example. Node-local burst buffers can also be dependent on the PFS (e.g., PLFS [5]) or are sometimes even managed directly by the PFS [49].

BurstFS [78], perhaps the most related work to ours, is a standalone burst buffer file system which does not require a centralized instance as well. However, GekkoFS is not limited to writing data locally like BurstFS. Instead, all data is distributed across all participating file system nodes to balance data workloads for write and read operations without sacrificing scalability. BeeOND [26] can create a job-temporal file system on a number of nodes similar to GekkoFS. BeeOND is, in contrast to our file system, POSIX compliant and our GekkoFS measurements show a much higher metadata throughput than offered by BeeOND [69, 71].

2.1.3 Metadata Scalability

The management of inodes (containing a file's metadata) and related directory blocks (containing data about which files belong to the directory) are the main scalability limitations of file systems in a distributed environment [73]. Typically, general-purpose PFSs distribute data across all available storage targets. While this technique works well for data, it does not achieve the same throughput when handling metadata [11, 54], although the file system community presented various techniques to tackle this challenge [5, 22, 50, 51, 79, 80]. The performance limitation can be attributed to the sequentialization enforced by underlying POSIX semantics which is particularly degrading throughput when an extremely large number of files is created in a single directory from multiple processes. This workload, common to HPC environments [5, 49, 50, 74], can become an even bigger challenge for upcoming data-science applications. GekkoFS handles directories and replaces directory entries by objects, stored within a strongly consistent key-value store which helps to achieve tens of millions of metadata operations for billions of files.

[1] IME: https://www.ddn.com/products/ime-flash-native-data-cache/.

[2] Datawarp: https://www.cray.com/datawarp.

www.dbooks.org

2.2 Design

In this section, we present goals, architecture, and general design of GekkoFS which allows scalable data and metadata performance. In general, any user without administrative access should be able to deploy GekkoFS. The user dictates on how many compute nodes and at which path the mountpoint of GekkoFS and its metadata and data is stored. The user is then presented with a single global namespace, consisting of the aggregated node-local storage of each node. To provide this functionality GekkoFS aims to achieve four core goals:

Scalability: GekkoFS should be able to scale with an arbitrary number of nodes and efficiently use available hardware.

Consistency model: GekkoFS should provide the same strong consistency as POSIX for common file system operations that access a specific data file. However, the consistency of directory operations, for example, can be relaxed.

Fast deployment: To avoid wasting valuable and expensive resources in HPC environments, the file system should startup within a minute and be ready for usage immediately by applications after the startup succeeds.

Hardware independence: GekkoFS should be able to support networking hardware that is commonly used in HPC environments, e.g., Omni-Path or Infiniband. The file system should be able to use the native networking protocols to efficiently move data between file system nodes. Finally, GekkoFS should work with modern and future storage technologies that are accessible to a user at an existing file system path.

2.2.1 POSIX Semantics

Similarly to PVFS [12] and OrangeFS [42], GekkoFS does not provide complex global locking mechanisms. In this sense, applications should be responsible to ensure that no conflicts occur, in particular, concerning overlapping file regions. However, the lack of distributed locking has consequences for operations where the number of affected file system objects is unknown beforehand, e.g., `readdir()` called by the `ls -l` command. In these *indirect file system operations*, GekkoFS does not guarantee to return the current state of the directory and follows the eventual-consistency model. Furthermore, each file system operation is synchronous without any form of caching to reduce file system complexity and to allow for an evaluation of its raw performance capabilities.

Further, GekkoFS does not support move or rename operations or linking functionality as HPC application studies have shown that these features are rarely or not used at all during the execution of a parallel job [37]. Such unsupported file system operations then trigger an I/O error to notify an application. Finally, security management in the form of access permissions is not maintained by GekkoFS since it already implicitly follows the security protocols of the node-local file system.

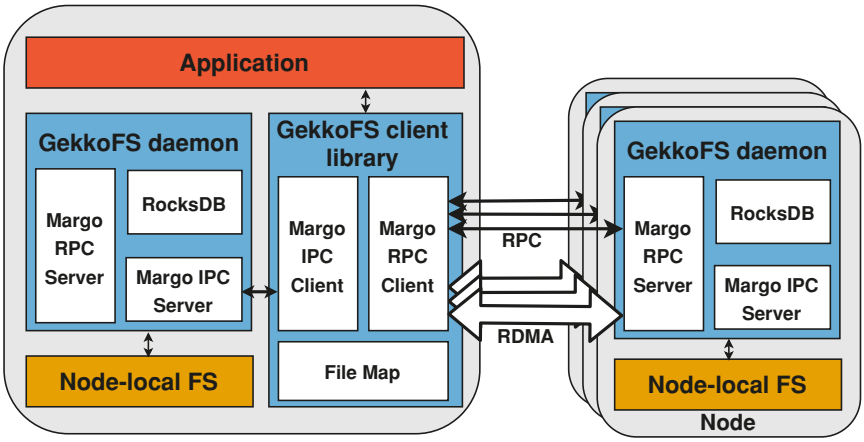


Fig. 1 GekkoFS architecture

2.2.2 Architecture

The architecture of GekkoFS (see Fig. 1) consists of two main components: a client library and a server process. An application that uses GekkoFS must first preload the client interposition library which intercepts all file system operations and forwards them to a server (*GekkoFS daemon*), if necessary. The GekkoFS daemon, which runs on each file system node, receives forwarded file system operations from clients and processes them independently, sending a response when finished. In the following paragraphs, we describe the client and daemon in more detail.

2.2.3 GekkoFS Client

The client consists of three components: (1) An interception interface that catches relevant calls to GekkoFS and forwards unrelated calls to the node-local file system; (2) a file map that manages the file descriptors of open files and directories, independently of the kernel; and (3) an RPC-based communication layer that forwards file system requests to local/remote GekkoFS daemons.

Each file system operation is forwarded via an RPC message to a specific daemon (determined by hashing of the file's path, similar to Lustre DNE 2[3]) where it is directly executed. In other words, GekkoFS uses a pseudo-random distribution to spread data and metadata across all nodes, also known as *wide-striping*. Because each client is able to independently resolve the responsible node for a file system operation, GekkoFS does not require central data structures that keep track of where metadata or data is located. To achieve a balanced data distribution for large files,

[3] https://lustre.ornl.gov/ecosystem-2016/documents/papers/LustreEco2016-Simmons-DNE.pdf.

www.dbooks.org

data requests are split into equally sized chunks before they are distributed across file system nodes (or GekkoFS daemons). The GekkoFS daemons then store each received chunk in a separate file (so-called *chunk files*) in its underlying node-local storage. If supported by the underlying network fabric protocol, the client exposes the relevant chunk memory region to the daemon, accessed via *remote-direct-memory-access* (RDMA).

2.2.4 GekkoFS Daemon

GekkoFS daemons consist of three parts: (1) A key-value store (KV store) used for storing metadata; (2) an I/O persistence layer that reads/writes data from/to the underlying local storage system; and (3) an RPC-based communication layer that accepts local and remote connections to handle file system operations.

Each daemon operates a single local RocksDB KV store [17]. RocksDB is optimized for NAND storage technologies with low latencies and fits GekkoFS' needs as SSDs are primarily used as node-local storage in today's HPC clusters. While RocksDB fits this use case well, the component is replaceable by other software or hardware solutions. Therefore, GekkoFS may introduce various choices for backends in the future to, for example, support recent key-value SSDs[4]

For the communication layer, we leverage on the *Mercury* RPC framework [62]. It allows GekkoFS to be network-independent and to efficiently transfer large data within the file system. Within GekkoFS, Mercury is interfaced indirectly through the *Margo* library which provides *Argobots*-aware wrappers to Mercury's API with the goal to provide a simple multi-threaded execution model [13, 58]. Using Margo allows GekkoFS daemons to minimize resource consumption of Margo's progress threads and handlers which accept and handle RPC requests [13].

Further, as indicated in Sect. 2.1.3, GekkoFS does not use a global locking manager. Therefore, when multiple processes write to the same file region concurrently, they may cause a shared write conflict with resulting undefined behavior with regards to which data is written to the underlying node-local storage. Such conflicts can, however, be handled locally by any GekkoFS daemon because it is using a POSIX-compliant node-local file system to store the corresponding data chunks, serializing access to the same chunk file. Note that such conflicts in a single file only affect one chunk at a time since the file's data is spread across many chunk files in the file system. As a result, chunks of that file are not disrupted during such a potential shared write conflict.

[4] https://www.samsung.com/semiconductor/global.semi.static/Samsung_Key_Value_SSD_enables_High_Performance_Scaling-0.pdf.

2.3 Evaluation

In this section, we evaluate the performance of GekkoFS based on various unmodified microbenchmarks which catch access patterns that are common in HPC applications. First, we describe the experimental setup and introduce the workloads that we simulate with microbenchmark applications. Then, we investigate the startup time of GekkoFS and compare metadata performance against a Lustre parallel file system. Although GekkoFS and Lustre have different goals, we point out the performances that can be gained by using GekkoFS as a burst buffer file system. Finally, we evaluate the data performance of GekkoFS and discuss the measured results.

2.3.1 Experimental Setup

We evaluated the performance of GekkoFS based on various unmodified microbenchmarks which catch access patterns that are common in HPC applications. Our experiments were conducted on the *MOGON II* supercomputer, located at the Johannes Gutenberg University Mainz in Germany. All experiments were performed on Intel 2630v4 Intel Broadwell processors (two sockets each). The node main memory capacity ranges from 64 GiB up to 512 GiB. MOGON II uses 100 Gbit/s Intel Omni-Path to establish a fat-tree network between all compute nodes. In addition, each node provides a data center Intel SATA SSD DC S3700 Series as scratch-space (*XFS* formatted) usable within a compute job. We used these SSDs for storing data and metadata of GekkoFS which uses an internal chunk size of 512 KiB. All Lustre experiments were performed on a Lustre scratch file system with 12 Object Storage Targets (OSTs), 2 Object Storage Servers (OSSs), and 1 Metadata Service (MDS) with a total of 1.2 PiB of storage.

Before each experiment iteration, GekkoFS daemons are restarted (requiring less than 20 s for 512 nodes), all SSD content is removed, and kernel buffer, inode, and dentry caches are flushed. The GekkoFS daemon and the application under test are pinned to separate processor sockets to ensure that file system and application do not interfere with each other.

2.3.2 Metadata Performance

We simulated common metadata intensive HPC workloads using the unmodified *mdtest* microbenchmark [41] to evaluate GekkoFS' metadata performance and compare it against a Lustre parallel file system. Although GekkoFS and Lustre have different goals, we point out the performances that can be gained by using GekkoFS as a burst buffer file system. In our experiments, mdtest performs *create*, *stat*, and *remove* operations in parallel in a single directory—an important workload in many

www.dbooks.org

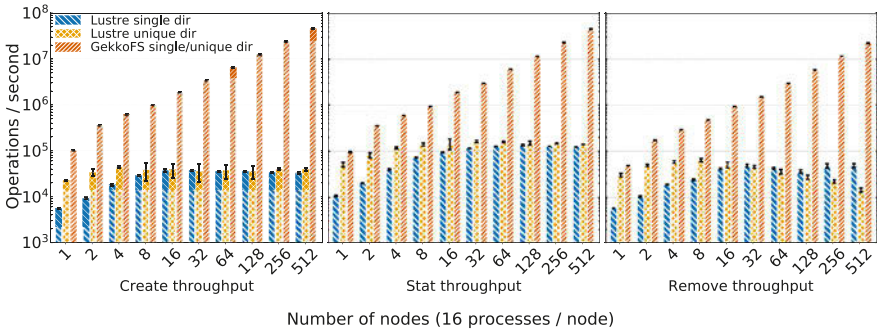


Fig. 2 GekkoFS' file create, stat, and remove throughput for an increasing number of nodes compared to a Lustre file system

HPC applications and among the most difficult workloads for a general-purpose PFS [74].

Each operation on GekkoFS was performed using 100,000 zero-byte files per process (16 processes per node). From the user application's perspective, all created files are stored within a single directory. However, due to GekkoFS' internally kept flat namespace, there is conceptually no difference in which directory files are created. This is in contrast to a traditional PFS that may perform better if the workload is distributed among many directories instead of in a single directory.

Figure 2 compares GekkoFS with Lustre in three scenarios with up to 512 nodes: file creation, file stat, and file removal. The y-axis depicts the corresponding operations per second that were achieved for a particular workload on a logarithmic scale. Each experiment was run at least five times with each data point representing the mean of all iterations. GekkoFS' workload scaled with 100,000 files per process, while Lustre's workload was fixed to four million files for all experiments. We fixed the number of files for Lustre's metadata experiments because Lustre was otherwise detecting hanging nodes when scaling to too many files.

Lustre experiments were run in two configurations: All processes operated in a single directory (`single dir`) or each process worked in its own directory (`unique dir`). Moreover, Lustre's metadata performance was evaluated while the system was accessible by other applications as well.

As seen in Fig. 2, GekkoFS outperforms Lustre by a large margin in all scenarios and shows close to linear scaling, regardless of whether Lustre processes operated in a single or in an isolated directory. Compared to Lustre, GekkoFS achieved around 46 million creates/s (∼1405×), 44 million stats/s (∼359×), and 22 million removes/s (∼453×) on 512 nodes. The standard deviation was less than 3.5% which was computed as the percentage of the mean. Therefore, we achieve our scalability goal, demonstrating the performance benefits of distributing metadata and decoupling directory entries from non-scalable directory blocks (see Sect. 2.2).

Additional GekkoFS experiments were also run while Mogon II was used by other users during production, revealing network interference within the cluster.

With up to 128 nodes we were unable to measure a difference in metadata operation throughput outside of the margin for error compared to the experiments in an undisturbed environment (see Fig. 2). For 256 and 512, we measured a reduced metadata operation throughput between 10 and 20% for create and stat operations. Remove operation throughput remained unaffected.

Lustre's metadata performance did not scale beyond approximately 32 nodes, demonstrating the aforementioned metadata scalability challenges in such a general-purpose PFS. Moreover, processes in Lustre experiments that operated within their own directory achieved a higher performance in most cases, except for the remove case where Lustre's `unique dir` remove throughput is reduced by over 70% at 512 nodes compared to Lustre's `single dir` throughput. This is because the time required to remove the directory of each process (in which it creates its workload) is included in the remove throughput and the number of created `unique` directories increases with the number of used processes in an experiment. Similarly, the time to create the process directories is also included in the create throughput but does not show similar behavior to the case of the remove throughput, indicating optimizations towards create operations.

2.3.3 Data Performance

We used the unmodified *IOR* [31] microbenchmark to evaluate GekkoFS' I/O performance for sequential and random access patterns in two scenarios: Each process is accessing its own file (file-per-process) and all processes access a single file (shared file). We used 8 KiB, 64 KiB, 1 MiB, and 64 MiB *transfer sizes* to assess the performances for many small I/O accesses and for few large I/O requests. We ran 16 processes on each client, each process writing and reading 4 GiB in total.

GekkoFS data performance is not compared with the Lustre scratch file system as the peak performance of the used Lustre partition, around 12 GiB/s, is already reached for $\leq$10 nodes for sequential I/O patterns. Moreover, Lustre has shown to scale linearly in larger deployments with more OSSs and OSTs being available [48].

Figure 3 shows GekkoFS' sequential I/O throughput in MiB/s, representing the mean of at least five iterations, for an increasing number of nodes for different transfer sizes. In addition, each data point is compared to the peak performance that all aggregated SSDs could deliver for a given node configuration, visualized as a white rectangle, indicating GekkoFS' SSD usage efficiency. In general, every result demonstrates GekkoFS' close to linear scalability, achieving about 141 GiB/s ($\sim$80% of the aggregated SSD peak bandwidth) and 204 GiB/s ($\sim$70% of the aggregated SSD peak bandwidth) for write and read operations for a transfer size of 64 MiB for 512 nodes.

Figure 4 shows GekkoFS' throughput for random accesses for an increasing number of nodes, showing close to linear scalability in all cases. The file system achieved up to 141 GiB/s write throughput and up to 204 GiB/s read throughput for 64 MiB transfer sizes at 512 nodes.

www.dbooks.org

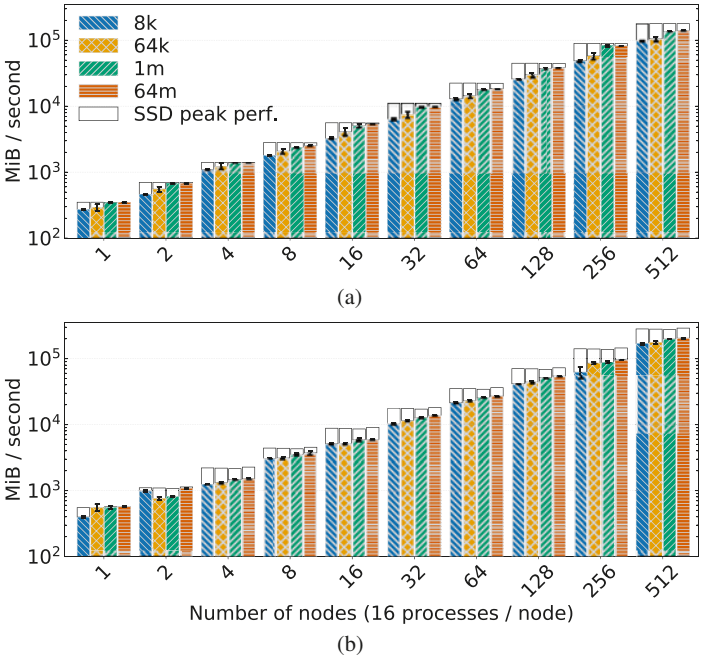


Fig. 3 GekkoFS' sequential throughput for each process operating on its own file compared to the plain SSD peak throughput. (**a**) Write throughput. (**b**) Read throughput

For the file-per-process cases, sequential and random access I/O throughput are similar for transfer sizes larger than the file system's chunk size (512 KiB). This is due to transfer sizes larger than the chunk size internally access whole chunk files while smaller transfer sizes access one chunk at a random offset. Consequently, random accesses for large transfer sizes are conceptually the same as sequential accesses. For smaller transfer sizes, e.g., 8 KiB, random write and read throughput decreased by approximately 33 and 60%, respectively, for 512 nodes owing to the resulting random access to positions within the chunks.

For the shared file cases, a drawback of GekkoFS' synchronous and cache-less design becomes visible. No more than approximately 150 K write operations per second were achieved. This was due to network contention on the daemon which maintains the shared file's metadata whose size needs to be constantly updated. To overcome this limitation, we added a rudimentary client cache to locally buffer size updates of a number of write operations before they are sent to the node that manages the file's metadata. As a result, shared file I/O throughput for sequential and random access were similar to file-per-process performances since chunk management on the daemon is then conceptually indifferent in both cases.

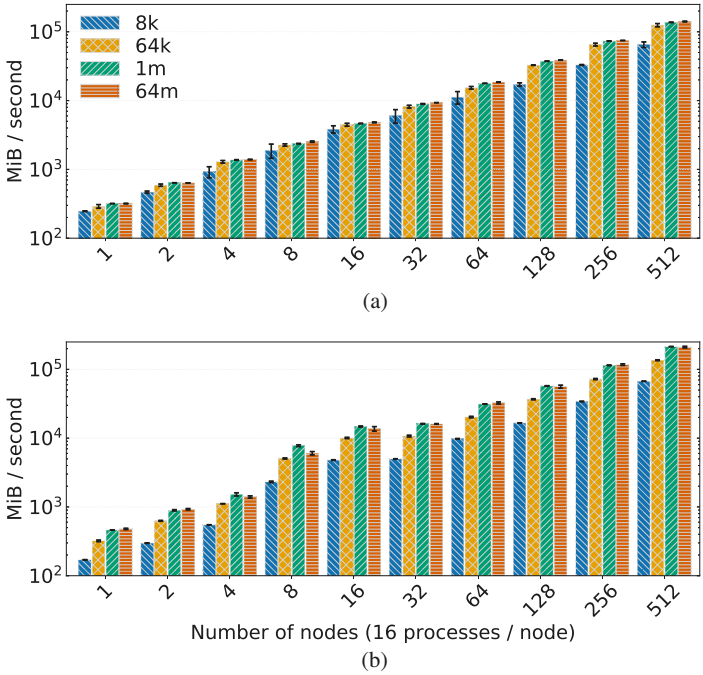


Fig. 4 GekkoFS' random throughput for each process operating on its own file. (**a**) Write throughput. (**b**) Read throughput

3 Scheduling and Deployment

In order to transfer the data to a previously generated on demand file system in time, the nodes that will be allocated to a job must be known in advance. Today's schedulers plan the resources of a supercomputer. The schedule is based on user requested wall times. Reality shows that the users requested wall times are very inaccurate, and thus the scheduler's predictions are unreliable.

Here two investigations were made and published. In the first work, we have shown that we can improve wall time estimates based on simple job metadata. We also used unconsidered metadata that is usually not publicly available [65].

Predicting the run times of jobs is only one aspect of the challenge. However, the essential factor is the prediction of node allocation to a job. In this second investigation, we have determined the influence of the wall-time on the node prediction [64]. The question we wanted to answer—How good do wall time predictions have to be to predict the allocated nodes accurately?

www.dbooks.org

3.1 Walltime Prediction

One of the challenges is to know which nodes are going to be allocated to a queued job. The HPC scheduler predicts these nodes based on the user given wall times. Therefore, we have decided to evaluate whether there is an easy way to predict such wall time automatically. Our proposed approach for wall time prediction is to train an individual model for every user. For this, we used methods from the machine learning domain and added job metadata, which was in previous work unconsidered. As historical data, we used workloads from two HPC-systems at the Karlsruhe Institute for Technology/Steinbuch Centre for Computing [66], the ForHLR I + II [34, 35] clusters. To train the model, we used automatic machine learning (AUTOML). AUTOML automates the process of hyperparameter optimization and selecting the correct model. We have chosen the auto ML library auto-sklearn [20], which is based on scikit-learn [9, 52].

In Fig. 5 the comparison of the user given wall times and the wall time prediction is shown. As a metric, the median absolute error (medAE) in hours is depicted as cumulative distribution. A model trained with AUTOML shows for 60% of the users a medAE of approximately 1 h on the ForHLR I and 1.4 h for the ForHLR II. The user estimations show a medAE deviation of about 7.4 h on both clusters. So we are able to reduce the median absolute deviation from 7.4 down to 1.4 h in average. Considering the fact that simple methods were used and no insight was provided into the job payload, this result is very good.

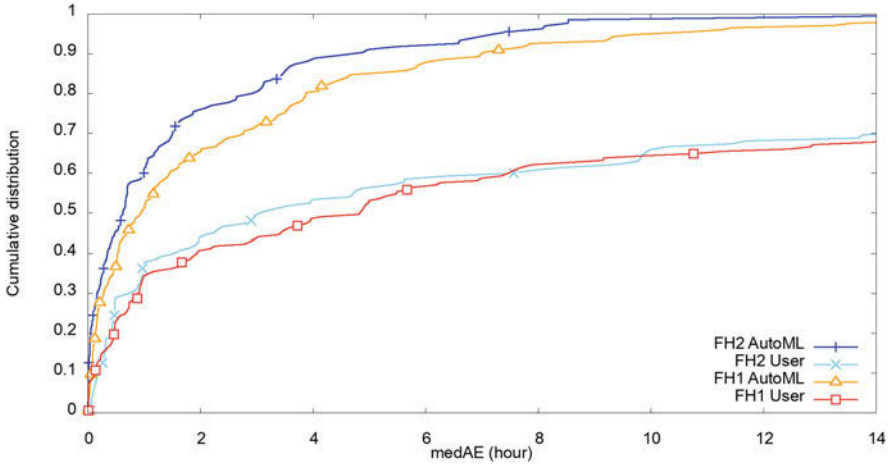


Fig. 5 Comparison of median absolute error (medAE) for ForHLR I+II. X-axis Median absolute error in hours, Y-Axis cumulative distribution

3.2 Node Prediction

As mentioned before predicting the run times of jobs is only one aspect of the challenge. However, the decisive factor is the accuracy of node allocation prediction. In this subsequent investigation, we have determined the impact on the node allocation accuracy with improved wall times. Therefore the ALEA Simulator [2] has been extended to simulate the time of the node allocation list [64].

We have conducted several simulations with subsequently improved job run time estimates, from inaccurate wall times as provided by users to fully accurate job run time estimates. For this purpose, we introduce $\tilde{T}_{\text{Req}}$, the "refined" requested wall time,

$$\tilde{T}_{\text{Req}} = T_{\text{Run}} + \lambda(T_{\text{Req}} - T_{\text{Run}}) \quad \text{with} \quad \lambda \in [0, 1], \tag{1}$$

where T_{Req} is the user requested wall time and T_{Run} is the run time of the job. To effectively simulate different precision of requested wall times, each job in the workload is modified by the same λ.

The result of the simulation is shown in Fig. 6, each bar represents a simulation with a different λ value. The bars are categorized into four groups based on the valid node allocation prediction (T_{NAP}). The blue part represents the jobs that are started immediately (instant) execution after the job is submitted to the batch system. These instantly started jobs offer of course no time to create a file system or even stage data. The orange part represents queued jobs with a T_{NAP} between 0 and 1 s. The green part shows jobs with a T_{NAP} from one second up to 10 min and red indicates long term prediction with a valid node allocation prediction over 10 min. The class of jobs with long-term predictions (red) is in our focus. This long-term predictions

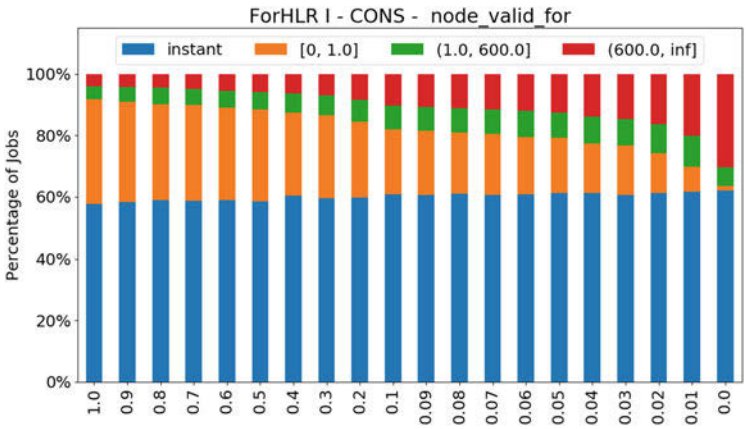


Fig. 6 Job distributions of ForHLR II workload with back-filling (CONS). Blue color denotes instant jobs, orange color means job having prediction $\leq$1 s, green color denotes jobs with 1 and 600 s and red color denotes long-term predictions($<$600 s)

www.dbooks.org

increase significantly only at very small $\lambda \leq 0.1$ which proves that very good run time estimates are needed.

3.3 On Demand Burst Buffer Plugin

From both evaluations, it is clear, that advanced data staging based on the scheduler prediction is not possible. Also, by using state-of-the-art methods such as machine learning, the accuracy is not sufficient. Therefore we decided to extend the functionality of the SLURM [15] scheduler. Slurm has a feature to manage burst buffers [16]. However, the current implementation status only includes support for the Cray DataWarp solution. Management of burst buffers using other storage technologies is documented, but not yet implemented. With the developed plugin, we extend the functionality of SLURM to create a file system on demand. For the prototype implementation, we also developed tools which deploy BeeOND (BeeGFS On Demand) as an on demand file system per job. Other parallel file systems,e.g. Lustre [7] or GekkoFS, can be added easily. The user requests an on demand file system by a job flag. He can also specify if data should be staged in and out. The SLURM controller marks the jobs and then does the corresponding operations [76].

3.4 Related Work

The requested wall times are unfortunately far away from the real used wall time. Gibbons [23, 24], and Downey [19] used historical workloads to predict the wall times of parallel applications. They predict wall times based on templates. These templates are created by analyzing previously collected metadata and grouped according to similarities. However, both approaches are restricted to simple definitions.

In the recent years, machine learning algorithms have been used to predict resource consumption in several studies [33, 40, 43, 44, 61, 70].

Predicting the run-time of jobs is also important in different topics, like for energy aware scheduling [3]. Here the applications' power and performance characteristics are considered to provide an optimized trade off between energy savings and job execution time.

However, all of the above mentioned studies do not try to evaluate the accuracy of the node allocation predictions. Most of the publications focus on observing the utilization of the HPC system and the reliability of the scheduler estimated job start times. In our work, we focus on the node allocation prediction and how good wall time estimates have to be. This directly affects, whether a cross-node, on demand, independent parallel FS can be deployed, and data can be pre-staged, or not.

4 Resource and Topology Detection

Compute nodes of modern HPC systems tend to get more heterogeneous. To plan a proper deployment of the GekkoFS file system on the compute nodes, knowledge of the underlying storage components are vital. This section describes what kind of resource information is of interest and shows possible ways to gather this information. Further, we discuss the architecture of the sysmap tool that we build to collect relevant information.

When thinking about the resources of a compute node, we distinguish between static and dynamic resource usage. Static resource information describes components that do not change frequently and are often similar between nodes. This includes the number of CPU cores, the amount of main memory, the number and capacity of node-local storage devices, or the type of file system. It is unlikely that this kind of hardware is replaced frequently. Otherwise different parts of a Cluster may have different configurations, e.g., one island of a Cluster may have more RAM than another island. On the other hand, dynamic resource usage describes the available resources at a certain point in time.

The goal is to hold a map of the resources available on a system. On the one hand, this can be used as an input for the data staging. On the other hand, such information is useful for the deployment of the file system. When the job scheduler has decided on which set of nodes a job will run, available hardware resources can be queried and an appropriate configuration to deploy the file system can be selected.

The sysmap tool can utilize existing hardware discovery libraries such as *hwloc* [8, 25] by using their interface. While *hwloc* does an excellent job for computing-related artifacts like the number of CPUs or cache sizes, it does not focus on the storage subsystem. Therefore, we use information from the `/proc` and `/sys` pseudo file systems to get information about the system. By reading the system configuration, the sysmap tool gathers information about partitions, mountpoints, file systems but also about available kernel modules and I/O-scheduler configuration. Moreover, we gather network information for InfiniBand networks by utilizing the well-known `ibnetdiscover` tool from the OFED distribution [47].

4.1 Design and Implementation

We have designed an extensible architecture for our sysmap tool. Each resource of interest is captured by a so-called *extractor*. Figure 7 shows a schematic UML-diagram of two extractors. Each extractor module consists of an abstract part, which defines the structure of the data that will be gathered and a specialized part which implements the logic to read the data from a specific source, by overriding the abstract interface. In Fig. 7, the `Filesystem_Extractor` and the `Disk_Extractor` are examples of the abstract parts. The `Linux::Filesystem_Extractor` and the

www.dbooks.org

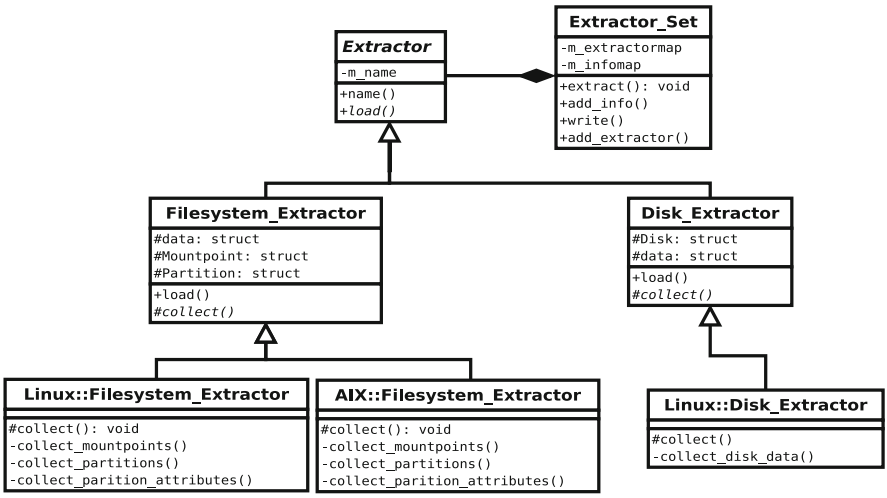


Fig. 7 Simple UML-Diagram for two example *extractor* modules of our system-map tool

`AIX::Filesystem_Extractor` are the specialized parts for extracting information of mountpoints and partitions of a specific system [46]. This is useful because the same information may be available on different systems through different sources. On the one hand, the user has to define the abstract extractor he wants, and the sysmap tool selects the source depending on what is available on the target system. On the other hand, we can implement specialized extractor modules for different sources resulting in an equivalent representation of the data for our tool. After gathering the data, the sysmap tool provides a wide variety of output formats presenting the data to the user. Since the tool is mentioned to be executed on multiple compute nodes, the recommended way is to store the results in a central database. Figure 8 depicts an overview of the general workflow of the resource discovery process. The sysmap tool runs on the compute nodes and gathers the resource information. Afterwards, the collected data is stored in a central resource database. For our working prototype, we use a *sqlite*[5] database. The information can be queried by the sysquery tool, which queries the resource database and outputs the selected data in JSON format. This way the querying component gets a machine-readable section of the required data which can be easy post-processed for they need. Further, the particular database query remains hidden from the user inside the sysquery tool. The datamodel of the resource database is shown in Fig. 9 and consists of four simple tables. The *HostTable* and *ExtractorTable* are to map the hostname or the extractor name to a numerical ID. Information extracted by an extractor is stored as JSON string in the *DataTable*. Further, a *DataID* is maintained to reference the data from an extractor. In the *Host2Data* table, the *DataID* is

[5] https://www.sqlite.org/index.html.

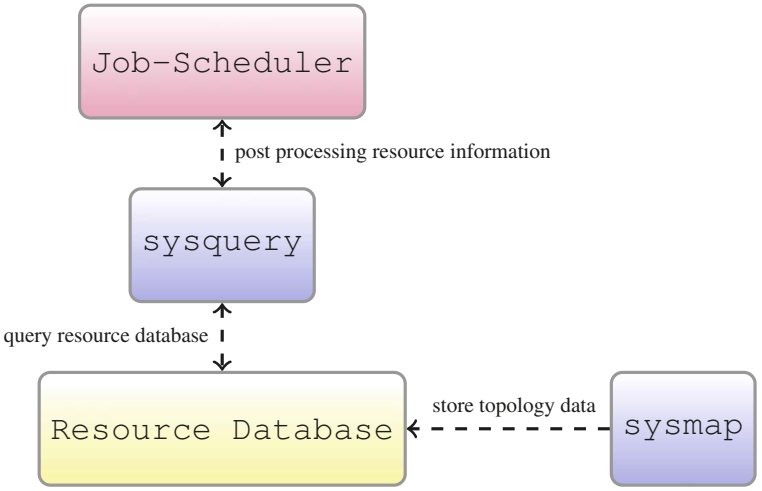


Fig. 8 Overview of resource discovery components, the blue components are part of the sysmap tool suite, the resource database is highlighted as the yellow box, the red box represents the querying component, in this case the Job-Scheduler

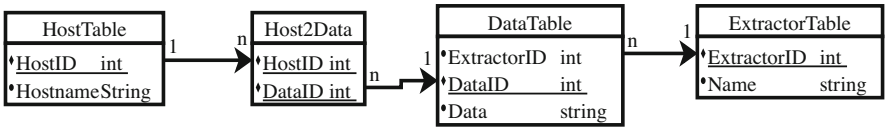


Fig. 9 The datamodel of the resource database

mapped with the corresponding *HostID*. This way, data that is equal across multiple nodes do not need to be stored multiple times but are easy to query. Since, the output of a query is a JSON string, it makes further processing and output easy for the calling script.

5 On Demand File System in HPC Environment

When using on demand file systems in HPC environments, the premise is that the normal operation should not be affected by the use of on demand file systems. The interference on other jobs should be avoided or even reduced. There should be also no modifications, that have a negative impact on the performance or utilization, of the system.

www.dbooks.org

5.1 Deploying on Demand File System

Usually HPC systems use a *batch system*, such as SLURM [60], MOAB [1], or LSF [30]. The batch system manages the resources of the cluster and starts the user jobs on allocated nodes. Before a job is started, a prologue script may be started on one or all allocated nodes and, if necessary, an epilogue script at the end of a job. These scripts are used to clean, prepare, or test the full functionality of the nodes. We modified these scripts to start the on demand file system upon request. During job submission, a user can request an on demand file system for the job. This solution has minimal impact on the HPC system operation. Users without the need for an on demand file system are not affected. An alternative way of deploying a on demand file system we have described in Sect. 3.3

5.2 Benchmarks

As initial benchmarks we tested the startup and shutdown time of the on demand file system (cf. Table 1). Comparing the startup time of BeeGFS on demand to the startup time of GekkoFS (512 Nodes under 20 s) it is clear, that BeeGFS takes too much time for startup and shutdown at larger scales. BeeGFS has a serial section in its startup where a status file is created on every node sequentially. This was also discussed on the mailing list [63] with a possible solution to improve the behavior in future releases.

In Fig. 10 we show the IoZone [10] benchmark to measure the read and write throughput of the on demand file system (solid line). The figure shows that performance increases linearly with the number of used compute nodes. The limiting factor here is the aggregate throughout of the used SATA-SSDs. A small throughput variation can be observed due to normal performance scattering of SSDs [36]. The dotted line indicates the theoretical throughput with NVMe devices. Here we assumed the performance for today's common PCIe $\times 4$ NVM devices [29] with a throughput of 3500/2000 MB/s of read/write performance.

In a further test, we evaluated the storage pooling feature of BeeGFS [4]. We created a storage pool for each switch according to the network topology.In other words, when writing to a storage pool, the data is distributed via the stripe count and chunk size, but remains within the storage pool and thus on a switch. Figure 11 shows the write throughput for three scenarios. Each scenario uses a different number of core switches with six being the full network capacity. In the first

Table 1 BeeGFS startup and shutwdown

Nodes	8	16	32	64	128	256
Startup (s)	10.2	16.7	29.3	56.5	152.1	222.4
Shutdown (s)	11.9	12.1	9.4	15.9	36.1	81.0

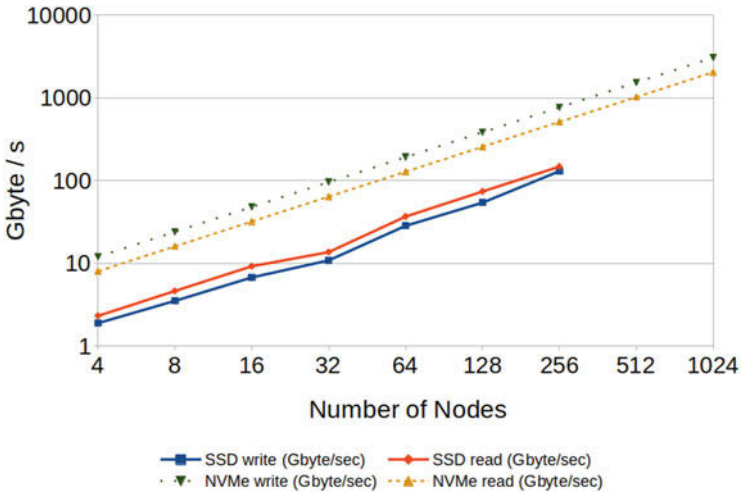


Fig. 10 Solid line: Read/write throughput. Dashed line: extrapolation with the theoretical peak of NVMe-SSDs



Fig. 11 IoZone write throughput with reduced number of core switches on 240 nodes

experiment, with all six core switches, there is only a minimal performance loss, which indicates a small overhead when using storage pools. In the second case we turned off three switches, and in the last case we turned off five switches. With reduced number of core switches the write throughput drops due to the reduced network capacity. If storage pools are created according to the topology, it is possible to achieve the same performance as with all six switches.

www.dbooks.org

5.3 Concurrent Data Staging

We also considered the case of copying data back to the PFS while an application is running. For this purpose, we evaluated NAStJA [6] with concurrent data staging. To stage the data, during the NAStJA execution, we used the parallel copy tool dcp [59]. The configuration for this use-case:

- We used 24 nodes with 20 cores per node.
- NAStJA was executed on 23 nodes with 20 tasks per node.
- BeeOND was started on all 24 nodes using the idle node as metadata server.
- Three different scenarios were evaluated during the application execution:
  - without data staging,
  - data staging using every node with one task per node for data staging,
  - data staging using the node, where only the meta-data server is running, with 4 tasks executed on this node.

Figure 12 shows the average execution time per time-step of five runs in our different scenarios. In the beginning, the slowdown is significant (orange line) due to the high amount of metadata operations. In this case, a portion of the data is indexed on every node. This indexing is causing interference with the application. When using only the MDS-server to copy the data (green line), the indexing is done only on the MDS-server.

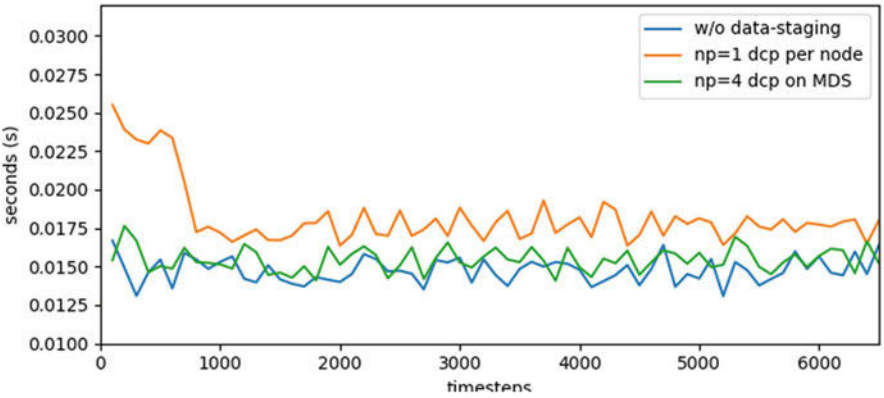


Fig. 12 Average execution time per time-step (5 runs). Without data staging(blue). Concurrent data staging using the meta-data node (green) and using every node (orange)

6 GekkoFS on NVME Based Storage Systems

Recently, new storage technologies such as NVME SSDs have been introduced to modern HPC systems. To evaluate the GekkoFS file system, for future systems, we performed some benchmarks using NVME SSDs. For demonstration we installed GekkoFS on the cluster Taurus [67] of TU Dresden. Taurus consists of ca. 47,000 cores of different architectures. For the demonstration, we use 8 NVME nodes of the HPC-DA [28] extension of Taurus. This extension consists of 90 nodes, and a single node has 8 Intel SSD DC P4610 Series NVME storage with 3.2 TB capacity and a peak bandwidth of 3.2 GB/s. Each node has 2 sockets Intel Xeon E5-2620 v4 with 32 cores and 64 GB main memory. Further, the NVME nodes are equipped with two 100 Gbit/s EDR Infiniband links with a peak bandwidth of 25 GB/s each. This experiment aims to investigate how well GekkoFS performs on new storage architectures.

We installed GekkoFS on Taurus using the Infiniband network provider. For our demonstration, we use 8 NVME nodes in this setup. The nodes are client and server in one. We assign one NVME card per node as backing storage to the GekkoFS daemon. This results in a distributed file system with a total capacity of 25.6 TB and a theoretical maximum bandwidth of $8 \times 3.2\,\text{GB/s} = 25.6\,\text{GB/s}$ for this configuration. To measure the data throughput of GekkoFS and investigate the impact of different access patterns to the file system we utilize the IOR benchmark.

We perform strong scaling tests with 8, 16, 32 and 64 processes writing and reading 1 TB of data. Therefore, we adjust the block size and transfer size for a different number of processes. To avoid interference, we pin the IOR processes to one socket while the GekkoFS daemon is pinned to the other. Before the creation of the GekkoFS file system the NVME devices were cleared, and a new Ext4 file system was created as an underlying file system on the block device. We measure different access patterns, file per process with sequential and random accesses and shared file with sequential access. To avoid measuring cache effects, we flush the page, inode and dentry caches of the operating system before each run.

Figure 13 shows the sequential access pattern. In the figure, one can see that the write bandwidth is stable at around 22 GB/s for all runs. The variation is small, and the values are near to the peak bandwidth of 25 GB/s for this setup. The suitable write bandwidths came from the relatively large transfer sizes of 64 MB to benefit from RDMA. For the read bandwidth, we get values between 13 and 17 GB/s. Also the read bandwidth first decreases when more processes are used and then increases again at the 64 processes. Such a poor read bandwidth is a behavior which could not be observed for the other measurements on MOGON II, where read and write bandwidth are almost equal, and is certainly a point of further investigation.

Figure 14 depicts the random access case. The results are similar to the sequential access pattern, which was expected because the internal handling of GekkoFS makes no difference for these cases. The write bandwidth is stable between 22 and 23 GB/s

www.dbooks.org

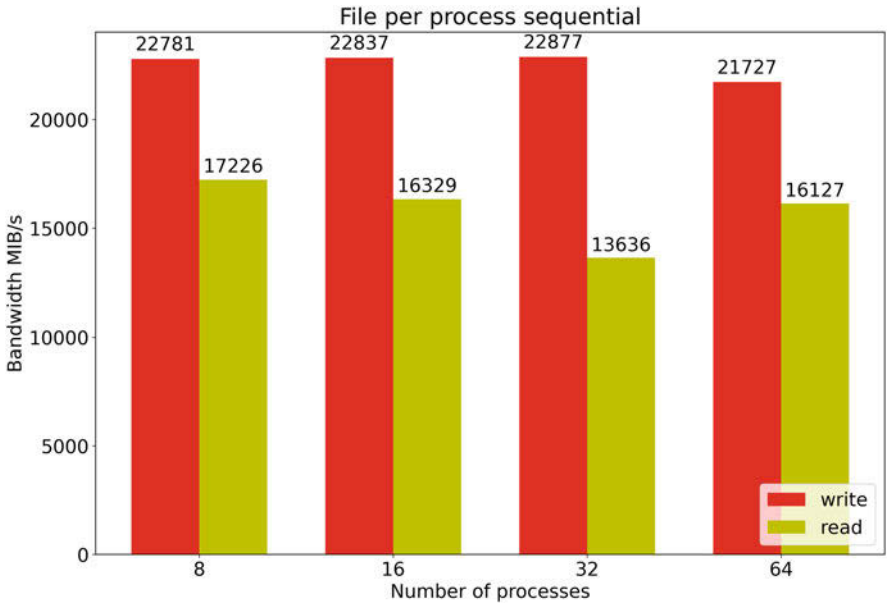


Fig. 13 IOR on GekkoFS on 8 NVME nodes performing a sequential file per process access pattern

and saturates the NVME SSD quite well. For the read, the achieved bandwidth is around 14 GB/s, the values are more stable than for the sequential case, which might be some cache effects.

In Fig. 15 we can see, that even for shared access pattern the results are similar to the file per process access pattern. The write bandwidth is again stable at 22 GB/s and the read bandwidth is around 16 GB/s except for the configuration with 32 processes where the read bandwidth is lower. This is also similar to the sequential file per process configuration in Fig. 13. As a result, we can see that GekkoFS can utilize NVME SSDs and is, therefore, ready for the next generation of storage systems. We could figure out that the different access patterns make no difference for the write bandwidth. For the read bandwidth, there is some bottleneck which needs further investigation. At the time of writing, multiple causes are imaginable; for example, the network layer for Infiniband might be an issue. This could also explain why this problem does not occur for the tests in Mainz because they have other network types.

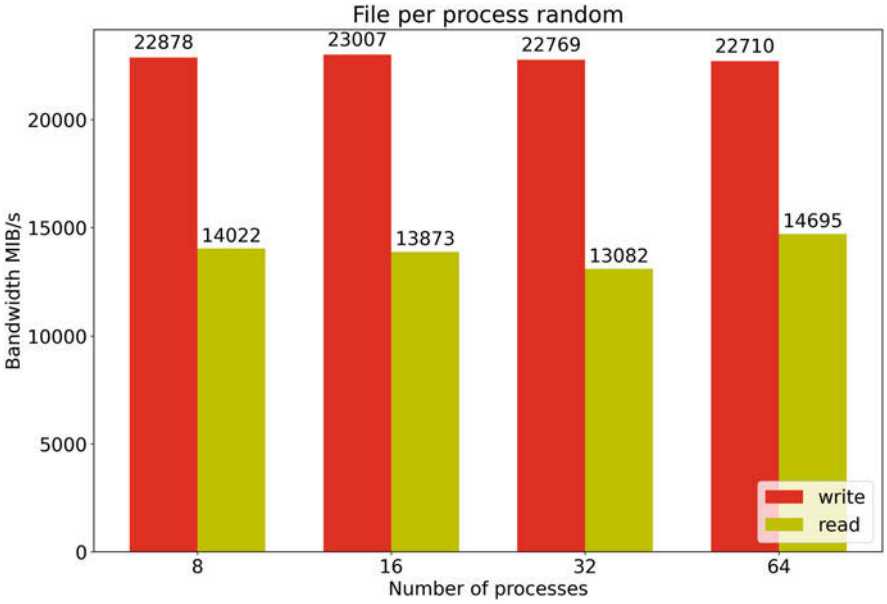


Fig. 14 IOR on GekkoFS on 8 NVME nodes performing a random file per process access pattern

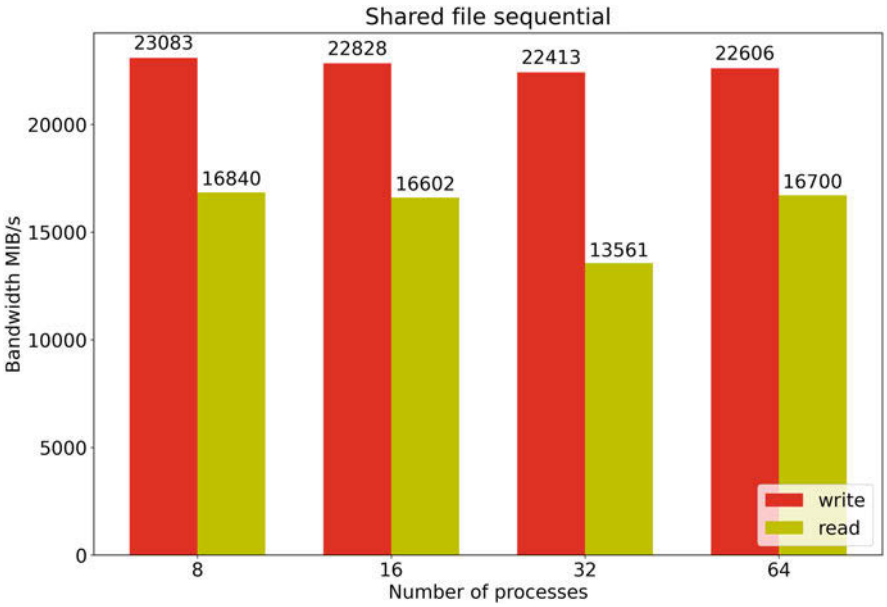


Fig. 15 IOR on GekkoFS on 8 NVME nodes performing a sequential shared file access pattern

www.dbooks.org

7 Conclusion

The goal of the ADA-FS project was to improve I/O performance for parallel applications. Therefore, a distributed burst buffer file system, and several components for deployment and data management were developed. The GekkoFS distributed burst buffer file system as the central part of the project was presented as a scalable and very flexible alternative to handle the challenging I/O patterns of scientific applications. Primarily through the innovative metadata management, it beats conservative shared parallel file systems for metadata intensive workloads by a margin. Thanks to its flexibility, GekkoFS offers the user an exclusive file system for his applications and eliminates several bottlenecks caused by the contention of a shared resource. In addition, GekkoFS has become a basis in the EU-funded *Next Generation I/O for Exascale* (NEXTGenIO) project where it will be continuously and collaboratively developed to support future storage technologies as well, such as persistent memory.

For successful data staging, investigations about the precision of the user-provided wallclock time of jobs were made. We could show how to improve wallclock estimates by considering the metadata of a job, and show a way to integrate the process of deployment and data staging into the job scheduler. Further, we present a tool suite to collect information about hardware resources of a compute node to support the deployment in a flexible manner.

Another topic that was not covered here is the analysis of the required POSIX semantics of parallel applications. These insights show during the design of the file system which operations are required to run scientific workloads. Further, its results can help the user to decide for a storage system that fits his needs best.

The evaluations showed that GekkoFS provides close to linear data and metadata scalability up to 512 nodes with tens of millions of metadata operations. Due to the decentralized and distributed design, the file system is set to be used in even larger environments as exascale environments are in close reach. Even on the latest storage infrastructure, GekkoFS can operate out of the box at the peak bandwidth at least for write operations.

Following this project, we plan further improvements on GekkoFS, for example, caching offers possibilities to gain even more performance. Another topic that we want to keep working on is the integration of GekkoFS into the job schedulers of the systems and the workflows of the user.

Conclusively, the project reached its goals by improving I/O performance of parallel applications, especially in the field of metadata intensive workloads where traditional parallel file systems are lacking performance.

Acknowledgments This work as part of the project ADA-FS is funded by the DFG Priority Program "Software for Exascale Computing" (SPPEXA, SPP 1648), which is gratefully acknowledged.

This research was conducted using the supercomputer ForHLR II and services offered by Karlsruher Insitute of Technology and the Steinbuch Centre for Computing. The authors gratefully acknowledge the time and granted access to ForHLR II.

Parts of this research were conducted using the supercomputer Mogon II and services offered by Johannes Gutenberg University Mainz. The authors gratefully acknowledge the computing time granted on Mogon II.

The authors gratefully acknowledge the GWK support for funding this project by providing computing time through the Center for Information Services and HPC (ZIH) at TU Dresden on the HPC-DA.

References

1. Adaptive Computing. http://www.adaptivecomputing.com
2. Alea 4: Job scheduling simulator (2019). https://github.com/aleasimulator
3. Auweter, A., Bode, A., Brehm, M., Brochard, L., Hammer, N., Huber, H., Panda, R., Thomas, F., Wilde, T.: A case study of energy aware scheduling on supermuc. In: Proceedings of the 29th International Conference on Supercomputing, ISC 2014, vol. 8488, pp. 394–409. Springer, New York (2014). https://doi.org/10.1007/978-3-319-07518-1_25
4. BeeGFS: BeeGFS Storage Pool. https://www.beegfs.io/wiki/StoragePools (2018). Accessed: August 1 201
5. Bent, J., Gibson, G.A., Grider, G., McClelland, B., Nowoczynski, P., Nunez, J., Polte, M., Wingate, M.: PLFS: a checkpoint filesystem for parallel applications. In: Proceedings of the ACM/IEEE Conference on High Performance Computing (SC), November 14–20, Portland, (2009)
6. Berghoff, M., Kondov, I., Hötzer, J.: Massively parallel stencil code solver with autonomous adaptive block distribution. IEEE Trans. Parallel Distrib. Syst. **29**(10), 2282–2296 (2018). https://doi.org/10.1109/TPDS.2018.2819672
7. Braam, P.J., Schwan, P.: Lustre: The intergalactic file system. In: Ottawa Linux Symposium, p. 50 (2002)
8. Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., Namyst, R.: hwloc: A generic framework for managing hardware affinities in HPC applications. In: Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, PDP 2010 (2010)
9. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., Varoquaux, G.: API design for machine learning software: experiences from the scikit-learn project. In: ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122 (2013)
10. Capps, D., Norcott, W.: Iozone filesystem benchmark (2008). http://iozone.org/
11. Carns, P., Yao, Y., Harms, K., Latham, R., Ross, R., Antypas, K.: Production I/O characterization on the Cray XE6. In: Proceedings of the Cray User Group meeting, vol. 2013 (2013)
12. Carns, P.H., III, W.B.L., Ross, R.B., Thakur, R.: PVFS: A parallel file system for Linux clusters. In: 4th Annual Linux Showcase and Conference 2000. Atlanta (2000)
13. Carns, P.H., Jenkins, J., Cranor, C.D., Atchley, S., Seo, S., Snyder, S., Ross, R.B.: Enabling NVM for data-intensive scientific services. In: 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads, INFLOW@OSDI 2016. Savannah (2016)
14. Crandall, P., Aydt, R.A., Chien, A.A., Reed, D.A.: Input/output characteristics of scalable parallel applications. In: Proceedings Supercomputing '95, San Diego, p. 59 (1995)
15. Documentation, S.: Slurm workload manager—overview. https://slurm.schedmd.com/overview.html
16. Documentation, S.: Slurm workload manager—-slurm burst buffer guide. https://slurm.schedmd.com/burst_buffer.html

www.dbooks.org

17. Dong, S., Callaghan, M., Galanis, L., Borthakur, D., Savor, T., Strum, M.: Optimizing space amplification in rocksdb. In: Proceedings of the 8th Biennial Conference on Innovative Data Systems Research, CIDR 2017. Chaminade (2017)
18. Dorier, M., Antoniu, G., Ross, R.B., Kimpe, D., Ibrahim, S.: Calciom: Mitigating I/O interference in HPC systems through cross-application coordination. In: Processing Symposium on 2014 IEEE 28th International Parallel and Distributed, pp. 155–164. Phoenix (2014)
19. Downey, A.B.: Predicting queue times on space-sharing parallel computers. In: Proceedings of the 11th International Parallel Processing Symposium, pp. 209–218. IEEE, Piscataway (1997)
20. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 28, pp. 2962–2970. Curran Associates, Red Hook (2015). http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf
21. Folk, M., Cheng, A., Yates, K.: HDF5: a file format and I/O library for high performance computing applications. In: Proceedings of the Supercomputing, vol. 99, pp. 5–33 (1999)
22. Frings, W., Wolf, F., Petkov, V.: Scalable massively parallel I/O to task-local files. In: Proceedings of the ACM/IEEE Conference on High Performance Computing (SC), Portland (2009)
23. Gibbons, R.: A historical application profiler for use by parallel schedulers. In: Job Scheduling Strategies for Parallel Processing, pp. 58–77. Springer, Berlin (1997)
24. Gibbons, R.: A historical profiler for use by parallel schedulers. Master's thesis, University of Toronto (1997)
25. Goglin, B.: Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc). In: 2014 International Conference on High Performance Computing and Simulation (HPCS), pp. 74–81. IEEE, Piscataway (2014)
26. Herold, F., Breuner, S., Heichler, J.: An introduction to beegfs (2014). https://www.beegfs.io/docs/whitepapers/Introduction_to_BeeGFS_by_ThinkParQ.pdf
27. Hey, T., Tansley, S., Tolle, K.M. (eds.): The Fourth Paradigm: Data-Intensive Scientific Discovery. Microsoft Research (2009)
28. HPC for Data Analytics (2019). https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/HPCDA. Accessed 25 July 2019
29. Hung, J.J., Bu, K., Sun, Z.L., Diao, J.T., Liu, J.B.: PCI express-based NVMe solid state disk. In: Applied Mechanics and Materials, vol. 464, pp. 365–368. Trans Tech Publications (2014)
30. IBM—platform computing. http://www.ibm.com/systems/platformcomputing/products/lsf/
31. Ior data benchmark (2018). https://github.com/hpc/ior
32. Jasak, H., Jemcov, A., Tukovic, Z., et al.: Openfoam: a C++ library for complex physics simulations. In: International Workshop on Coupled Methods in Numerical Dynamics, vol. 1000, pp. 1–20 (2007)
33. Kapadia, N.H., Fortes, J.A.: On the design of a demand-based network-computing system: The purdue university network-computing hubs. In: Proceedings of the Seventh International Symposium on High Performance Distributed Computing, pp. 71–80. IEEE, Piscataway (1998)
34. Forschungshochleistungsrechner ForHLR 1 (2018). www.scc.kit.edu/dienste/forhlr1.php
35. Forschungshochleistungsrechner ForHLR 2 (2018). www.scc.kit.edu/dienste/forhlr2.php
36. Kim, E.: SSD performance-a primer. In: Solid State Storage Initiative (2013)
37. Lensing, P.H., Cortes, T., Hughes, J., Brinkmann, A.: File system scalability with highly decentralized metadata on independent storage devices. In: IEEE/ACM 16th International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, pp. 366–375 (2016)
38. Liu, N., Cope, J., Carns, P.H., Carothers, C.D., Ross, R.B., Grider, G., Crume, A., Maltzahn, C.: On the role of burst buffers in leadership-class storage systems. In: IEEE 28th Symposium on Mass Storage Systems and Technologies, MSST 2012, Pacific Grove, pp. 1–11 (2012)
39. Lofstead, J.F., Klasky, S., Schwan, K., Podhorszki, N., Jin, C.: Flexible IO and integration for scientific codes through the adaptable IO system (ADIOS). In: Proceedings of the Sixth International Workshop on Challenges of Large Applications in Distributed Environments, CLADE@HPDC 2008, Boston, pp. 15–24 (2008)

40. Matsunaga, A., Fortes, J.A.: On the use of machine learning to predict the time and resources consumed by applications. In: Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing, pp. 495–504. IEEE Computer Society, Washington (2010)
41. Mdtest metadata benchmark (2018). https://github.com/hpc/ior
42. Moore, M., Bonnie, D., Ligon, B., Marshall, M., Ligon, W., Mills, N., Quarles, E., Sampson, S., Yang, S., Wilson, B.: Orangefs: Advancing PVFs. FAST poster session (2011)
43. Mu'alem, A.W., Feitelson, D.G.: Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distrib. Syst. **12**(6), 529–543 (2001)
44. Nadeem, F., Fahringer, T.: Using templates to predict execution time of scientific workflow applications in the grid. In: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, pp. 316–323. IEEE Computer Society, Washington (2009)
45. Nieuwejaar, N., Kotz, D., Purakayastha, A., Ellis, C.S., Best, M.L.: File-access characteristics of parallel scientific workloads. IEEE Trans. Parallel Distrib. Syst. **7**(10), 1075–1089 (1996)
46. Oeste, S., Kluge, M., Soysal, M., Streit, A., Vef, M., Brinkmann, A.: Exploring opportunities for job-temporal file systems with ADA-FS. In: Proceedings of the First Joint International Workshop on Parallel Data Storage and Data Intensive Scalable Computing Systems (2016)
47. OpenFabrics Alliance: Open fabric enterprise distribution (2016). https://www.openfabrics.org/. Accessed 16 July 2016
48. Oral, S., Dillow, D.A., Fuller, D., Hill, J., Leverman, D., Vazhkudai, S.S., Wang, F., Kim, Y., Rogers, J., Simmons, J., et al.: OLCFs 1 TB/s, next-generation lustre file system. In: Proceedings of the Cray User Group Conference (CUG 2013), pp. 1–12 (2013)
49. Oral, S., Shah, G.: Spectrum scale enhancements for coral. Paper presentation slides at supercomputing'16. (2016). http://files.gpfsug.org/presentations/2016/SC16/11_Sarp_Oral_Gautam_Shah_Spectrum_Scale_Enhancements_for_CORAL_v2.pdf
50. Patil, S., Gibson, G.A.: Scale and concurrency of GIGA+: file system directories with millions of files. In: Proceedings of the Ninth USENIX Conference on File and Storage Technologies, San Jose, pp. 177–190 (2011)
51. Patil, S., Ren, K., Gibson, G.: A case for scaling HPC metadata performance through de-specialization. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis, Salt Lake City, pp. 30–35 (2012)
52. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
53. Qian, Y., Li, X., Ihara, S., Zeng, L., Kaiser, J., Süß, T., Brinkmann, A.: A configurable rule based classful token bucket filter network request scheduler for the lustre file system. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC), Denver, pp. 6:1–6:12 (2017)
54. Ren, K., Zheng, Q., Patil, S., Gibson, G.A.: IndexFS: scaling file system metadata performance with stateless caching and bulk insertion. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2014, New Orleans, pp. 237–248 (2014)
55. Ross, R., Thakur, R., Choudhary, A.: Achievements and challenges for I/O in computational science. In: Journal of Physics: Conference Series, vol. 16, p. 501 (2005)
56. Ross, R.B., Latham, R.: PVFS–PVFS: a parallel file system. In: Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, Tampa, p. 34 (2006)
57. Schmuck, F.B., Haskin, R.L.: GPFS: a shared-disk file system for large computing clusters. In: Proceedings of the FAST '02 Conference on File and Storage Technologies, Monterey, pp. 231–244 (2002)
58. Seo, S., Amer, A., Balaji, P., Bordage, C., Bosilca, G., Brooks, A., Carns, P.H., Castelló, A., Genet, D., Hérault, T., Iwasaki, S., Jindal, P., Kalé, L.V., Krishnamoorthy, S., Lifflander, J., Lu, H., Meneses, E., Snir, M., Sun, Y., Taura, K., Beckman, P.H.: Argobots: A lightweight low-level threading and tasking framework. IEEE Trans. Parallel Distrib. Syst. **29**(3), 512–526 (2018)

www.dbooks.org

59. Sikich, D., Di Natale, G., LeGendre, M., Moody, A.: mpiFileUtils: A Parallel and Distributed Toolset for Managing Large Datasets. Tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore (2017)
60. Slurm—schedmd. http://www.schedmd.com
61. Smith, W.: Prediction services for distributed computing. In: IEEE International Parallel and Distributed Processing Symposium, pp. 1–10. IEEE, Piscataway (2007)
62. Soumagne, J., Kimpe, D., Zounmevo, J.A., Chaarawi, M., Koziol, Q., Afsahi, A., Ross, R.B.: Mercury: enabling remote procedure call for high-performance computing. In: 2013 IEEE International Conference on Cluster Computing, CLUSTER 2013, Indianapolis, pp. 1–8 (2013)
63. Soysal, M.: fhfgs mailing list discussion (2017). https://groups.google.com/d/msg/fhgfs-user/g8ysFS35Ucs/E-RtxKyiCAAJ
64. Soysal, M., Berghoff, M., Klusáček, D., Streit, A.: On the quality of wall time estimates for resource allocation prediction. In: Proceedings of the 48th International Conference on Parallel Processing: Workshops, ICPP 2019, pp. 23:1–23:8. ACM, New York, (2019). https://doi.org/10.1145/3339186.3339204
65. Soysal, M., Berghoff, M., Streit, A.: Analysis of job metadata for enhanced wall time prediction. In: Proceedings of the 22nd International Workshop on Job Scheduling Strategies for Parallel Processing, JSSPP 2018, Vancouver. Revised selected papers, Klusáček, D (ed.) Lecture Notes in Computer Science, vol. 11332, p. 14 S. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-10632-4_1. 46.11.01; LK 01
66. Steinbuch Center for Computing (SCC). http://www.scc.kit.edu (2016). Accessed 16 August 2016
67. Bull HPC-Cluster Taurus (2019). https://doc.zih.tu-dresden.de/hpc-wiki/bin/view/Compendium/SystemTaurus. Accessed 25 July 2019
68. Thapaliya, S., Bangalore, P., Lofstead, J.F., Mohror, K., Moody, A.: Managing I/O interference in a shared burst buffer system. In: 45th International Conference on Parallel Processing, ICPP 2016, Philadelphia, pp. 416–425 (2016)
69. Thinkparq, BeeGFS: Beegfs the leading parallel cluster file system (2018). https://www.beegfs.io/docs/BeeGFS_Flyer.pdf
70. Tsafrir, D., Etsion, Y., Feitelson, D.G.: Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distrib. Syst. **18**(6), 789–803 (2007)
71. Vef, M., Moti, N., Süß, T., Tacke, M., Tocci, T., Nou, R., Miranda, A., Cortes, T., Brinkmann, A.: Gekkofs—a temporary burst buffer file system for HPC applications. J. Comput. Sci. Technol. **35**(1), 72–91 (2020)
72. Vef, M., Moti, N., Süß, T., Tocci, T., Nou, R., Miranda, A., Cortes, T., Brinkmann, A.: Gekkofs—a temporary distributed file system for HPC applications. In: IEEE International Conference on Cluster Computing, CLUSTER 2018, Belfast, pp. 319–324 (2018)
73. Vef, M.A.: Analyzing file create performance in IBM spectrum scale. Master's thesis, Johannes Gutenberg University Mainz (2016). http://www.staff.uni-mainz.de/vef/pubs/vef2016thesis.pdf
74. Vef, M.A., Tarasov, V., Hildebrand, D., Brinkmann, A.: Challenges and solutions for tracing storage systems: a case study with spectrum scale. ACM Trans. Storage **14**(2), 18:1–18:24 (2018)
75. Vilayannur, M., Nath, P., Sivasubramaniam, A.: Providing tunable consistency for a parallel file store. In: Proceedings of the FAST '05 Conference on File and Storage Technologies, San Francisco (2005)
76. Voigt, V.: Entwicklung eines on-demand burst-buffer-plugins fuer HPC-batch-systeme
77. Wang, F., Xin, Q., Hong, B., Brandt, S.A., Miller, E., Long, D., McLarty, T.: File system workload analysis for large scale scientific computing applications. Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore (2004)
78. Wang, T., Mohror, K., Moody, A., Sato, K., Yu, W.: An ephemeral burst-buffer file system for scientific applications. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2016, Salt Lake City, pp. 807–818 (2016)

79. Xing, J., Xiong, J., Sun, N., Ma, J.: Adaptive and scalable metadata management to support a trillion files. In: Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2009, Portland (2009)
80. Yang, S., Ligon III, W.B., Quarles, E.C.: Scalable distributed directory implementation on orange file system. In: Proceedings of the IEEE International Workshop on Storage Network Architecture and Parallel I/Os (SNAPI) (2011)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

AIMES: Advanced Computation and I/O Methods for Earth-System Simulations



Julian Kunkel, Nabeeh Jumah, Anastasiia Novikova, Thomas Ludwig, Hisashi Yashiro, Naoya Maruyama, Mohamed Wahib, and John Thuburn

Abstract Dealing with extreme scale earth system models is challenging from the computer science perspective, as the required computing power and storage capacity are steadily increasing. Scientists perform runs with growing resolution or aggregate results from many similar smaller-scale runs with slightly different initial conditions (the so-called ensemble runs). In the fifth Coupled Model Intercomparison Project (CMIP5), the produced datasets require more than three Petabytes of storage and the compute and storage requirements are increasing significantly for CMIP6. Climate scientists across the globe are developing next-generation models based on improved numerical formulation leading to grids that are discretized in alternative forms such as an icosahedral (geodesic) grid. The developers of these models face similar problems in scaling, maintaining and optimizing code. Performance portability and the maintainability of code are key concerns of scientists as, compared to industry projects, model code is continuously revised and extended to incorporate further levels of detail. This leads to a rapidly growing code base that is rarely refactored. However, code modernization is important to maintain productivity

J. Kunkel (✉)
University of Reading, Reading, England
e-mail: j.m.kunkel@reading.ac.uk

N. Jumah
Universität Hamburg, Hamburg, Germany

A. Novikova
Fraunhofer IGD, Rostock, Germany

T. Ludwig
Deutsches Klimarechenzentrum (DKRZ), Hamburg, Germany

H. Yashiro · N. Maruyama
RIKEN, Tokyo, Japan

M. Wahib
AIST Tokyo, Tokyo, Japan

J. Thuburn
University of Exeter, Exeter, England

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_5

of the scientist working with the code and for utilizing performance provided by modern and future architectures. The need for performance optimization is motivated by the evolution of the parallel architecture landscape from homogeneous flat machines to heterogeneous combinations of processors with deep memory hierarchy. Notably, the rise of many-core, throughput-oriented accelerators, such as GPUs, requires non-trivial code changes at minimum and, even worse, may necessitate a substantial rewrite of the existing codebase. At the same time, the code complexity increases the difficulty for computer scientists and vendors to understand and optimize the code for a given system. Storing the products of climate predictions requires a large storage and archival system which is expensive. Often, scientists restrict the number of scientific variables and write interval to keep the costs balanced. Compression algorithms can reduce the costs significantly but can also increase the scientific yield of simulation runs. In the AIMES project, we addressed the key issues of programmability, computational efficiency and I/O limitations that are common in next-generation icosahedral earth-system models. The project focused on the separation of concerns between domain scientist, computational scientists, and computer scientists.The key outcomes of the project described in this article are the design of a model-independent Domain-Specific Language (DSL) to formulate scientific codes that can then be mapped to architecture specific code and the integration of a compression library for lossy compression schemes that allow scientists to specify the acceptable level of loss in precision according to various metrics. Additional research covered the exploration of third-party DSL solutions and the development of joint benchmarks (mini-applications) that represent the icosahedral models. The resulting prototypes were run on several architectures at different data centers.

1 Introduction

The problems on the frontier of science requires extreme computational resources and data volumes across the disciplines. Examples of processes include the understanding of the earth mantle [10], plasma fusion [24], properties of steel [5], and the simulation of weather and climate. The simulation of weather and climate requires to model many physical processes such as the influence of radiation from the sun and the transport of air and water in atmosphere and ocean [8]. As these processes are complex, scientists from different fields collaborate to develop models for climate and weather simulations.

The mathematical model of such processes is discretized and encoded as computer model using numerical methods [53]. Different numerical methods can be used to approximate the mathematical models. A range of different numerical methods are used, including finite differences, finite volumes, and finite elements. All of these methods partition the domain of interest into small regions and apply stencil computations to approximate operations such as derivatives.

www.dbooks.org

The necessary computations include variables (fields) like temperature or pressure distributed spatially over some surface or space—the problem domain. Simple techniques divide a surface into rectangular smaller regions covering the whole domain. Such rectangular grids have a simple regular structure. Those grids fit computations well as the grid structure simply corresponds to the array notation of the programming languages. However, applying this grid to the globe leads to variable sizes of grid cells, e.g., the equator region has a coarse grid while the polar regions are a singularity. With such a shortcoming, rectangular grids are well suited for regional models but not for a global model.

Therefore, recent models targeting global simulations are developed using different grids. Moving to such alternative grids allows to solve the cell area problem for global models, but the formulation of the models is more complicated. Icosahedral grids are examples of such alternatives. An icosahedral grid results from projecting an icosahedron onto the surface of the globe. The surface of the globe is then divided into twenty spherical triangles with equal areas. Grid refinement is achieved with recursive division of the spherical arcs into halves. The resulting points of the division form four smaller spherical triangles within each spherical triangle. Such refinement is repeated until the needed resolution is reached. Icosahedral grids can be used with the triangles as the basic cell, but also hexagons can be synthesized.

Icosahedral grids have approximately uniform cell area and can be used for global models avoiding the cell area differences in contrast to the rectangular grids. However, complications arise when thinking of the technical side, where we need to know how to map the field data into data structures. Such technical details are challenging with the performance demand for the models.

The values of a field in the simulation is localized with respect to the grid cell depending on the numerical formulation. In one method, values of a field are localized at the centers of the cells—this can be a single value or multiple values with higher order methods. However, other methods localize values on the vertices, while others reside on the edges separating the cells (see Fig. 1). How the cells are connected to each other, i.e., the neighbors and orientation of the cells, is defined in the connectivity. A problem domain can be organized in a regular

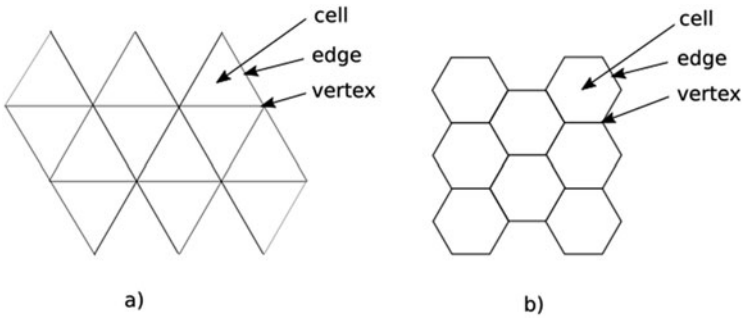


Fig. 1 Icosahedral grids and variables. (**a**) Triangular grid. (**b**) Hexagonal grid

fashion into so-called structured grids—following an easy schema to identify (left, right, ...) neighbors. Unstructured grids can follow a complex connectivity, e.g., using smaller and larger cells, or covering complex surfaces but require to store the connectivity information explicitly. The modern models explore both schemes due to their benefit; for instance, the structured grid can utilize compiler optimizations more effectively, while unstructured grids allow local refinement around areas of interest.

General-purpose languages (GPL), e.g., Fortran, are widely used to encode the discretized computer model. The simulation of the earth with a high resolution like $1\,km^2$ that is necessary to cover many smaller-scale physical processes, requires a huge computation effort and likewise storage capacity to preserve the results for later analysis. Due to uncertainty, scientists run a single experiment many times, multiplying the demand for compute and storage resources. Thus, the optimization of the codes for different architectures and efficiency is of prime importance to enable the simulations.

With existing solutions, scientists rewrite some code sections repeatedly with different optimization techniques that utilize the capabilities of the different machines. Hence, scientists must learn different optimization techniques for different architectures. The code duplication brings new issues and complexities concerning the development and the maintainability of the code.

Thus, the effort from the maintainers and developers of the models, who are normally scientists and not computer scientists, is substantial. Scientists' productivity is an important point to consider as they do activities that should not be their focus. Maintaining model codes throughout the lifecycle of the model is a demanding effort under all the mentioned challenges.

The structure of the icosahedral grids brings complications not only to the computation, but also to the storage of the field data. In contrast to regular grids, where multi-dimensional array notation fits to hold the data, icosahedral grids do not necessarily map directly to simple data structures. Besides the challenge of file format support, modern models generate large amounts of data that impose pressure on storage systems. Recent models are developed with higher-resolution grids, and include more fields and processes. Simulations writing terabytes of data to the storage system push towards optimizing the use of the storage by applying data-reduction techniques.

The development of simulation models unfolds many challenges for the scientific community. Relevant challenges for this project are:

- **Long life**: The lifecycle of earth system models is long in comparison to the turnover of the computer technology mainly in terms of processor architectures.
- **Performance and efficiency**: The need for performance and the optimal use of the hardware resources is an important issue.
- **Performance-portability**: Models are run on different machines and on different architectures. They must use the available capabilities effectively.

www.dbooks.org

- **Collaboration**: Another point is the collaborative efforts to develop models—involving PhD students to contribute pieces of science code to a large software project—that complicates the maintenance and software engineering of the models.
- **Data volume**: the large amounts of data must be handled efficiently.

1.1 The AIMES Project

To address challenges facing earth system modeling, especially for icosahedral models, the project *Advanced Computation and I/O Methods for Earth-System Simulations* (AIMES) investigated approaches to mitigate the aforementioned programming and storage challenges.

The AIMES project is part of the SPPEXA program and consisted of the consortium:

- Universität Hamburg, Germany
- Institut Pierre Simon Laplace (IPSL), Université Versailles Saint-Quentin-en-Yvelines, France
- RIKEN Advanced Institute for Computational Science, Japan
- Global Scientific Information and Computing Center, Tokyo Institute of Technology, Japan

The project started in March 2016 with plans for 3 years.

The main objectives of the project were: (1) enhance programmability; (2) increase storage efficiency; (3) provide a common benchmark for icosahedral models. The project was organized in three work packages covering these aspects and a supplementary project management work package to achieve the three project objectives. The strategy of the work packages is layed out in the following.

Under the first work package, higher-level scientific concepts are used to develop a dialect for each of three icosahedral models: DYNAMICO [17], ICON [54], and NICAM [47]. A domain-specific language (DSL) is the result of finding commonalities in the three dialects. Also, a light weight source-to-source translation tool is developed. Targeting different architectures to run the high-level code is an important aspect of the translation process. Optimizations include applying parallelization to the different architectures. Also, providing a memory layout that fits the different architectures is considered.

Under the second work package, data formats for icosahedral models are investigated to deal with the I/O limitations. Lossy compression methods are developed to increase storage efficiency for icosahedral models. The compression is guided with user-provided configuration to allow to use suitable compression according to the required data properties.

Under the third work package, relevant kernels are selected from the three icosahedral models. A mini-IGCM is developed based on each of the three models

to offer a benchmark for icosahedral models. Developed code is used to evaluate the DSL and the compression of icosahedral global modeling.

The **key outcomes** of the project described in this article are: (1) the design of an effective light-weight DSL that is able to abstract the scientific formulation from architecture-specific capabilities; (2) the development of a compression library that separates the specification of various qualities that define the tolerable error of data from the implementation. We provide some further large-scale results of our compression library for large scale runs extending our papers about the library [36, 37]; (3) the development of benchmarks for the icosahedral models that are mini-applications of the models.

Various additional contributions were made that are summarized briefly with citations to the respective papers:

- We researched the impact of lossless data compression on energy consumption in [2]. In general, the energy consumption increases with the computational intensity of the compression algorithm. However, there are algorithms that are efficient and less computational intense that can improve the energy-efficiency.
- We researched compilation time of code that is generated from the DSL using alternative optimization options on different compilers [20]. Different optimization options for different files allow different levels of performance, however, compilation time is also an important point to consider. Results show that some files need less optimization focus while others need further care. Small performance drops are measured with considerable reduction in compile times when the suitable compilation options are chosen.
- We researched annotating code for instrumentation automatically by our translation tool, to identify resource consuming kernels [21]. Instrumentation allows to better find where to focus the optimization efforts. We used the DSL translation tool to annotate kernels and make generated code ready for instrumentation. As a result performance measurements were recorded with reduced effort as manual preparations are not needed anymore.
- We researched applying vector folding to icosahedral grid codes in a bachelor thesis [49]. Vector folding allows to improve use of caches by structuring data in a way accounting for caches and data dimensionality. Results show that vector folding was difficult to apply manually to icosahedral grids. Performance was raised but not significantly as a result of the needed effort that should be invested to rewrite kernels with this kind of optimization.
- We involved ASUCA and the use of Hybrid Fortran [43] to port original CPU code to GPUs, to look at a different model with different requirements.

This article is structured as follows: first, the scope of the state-of-the-art and related work is sketched in Sect. 2. In Sect. 3, an alternative development approach for code-modernization is introduced. Various experiments to evaluate the benefit of the approach are shown in Sect. 4. The compression strategy is described in Sect. 5 and evaluated in Sect. 6. The benchmarks for the icosahedral models are discussed in Sect. 7. Finally, the article is concluded in Sect. 8.

www.dbooks.org

2 Related Work

The related work covers research closely related to domain-specific languages in climate and weather and the scientific data compression.

2.1 Domain-Specific Languages

DSLs represent an important approach to provide performance portability and support model development. A DSL is always developed having a particular domain in mind. Some approaches support multiple layers of abstraction. A high-level abstraction for the finite element method is provided with Firedrake [44]. The ExaStencils pipeline generally addresses stencil codes and their operations [18, 33] and many research works introduce sophisticated schemes for the optimization of stencils [7, 9].

One of the first DSLs which were developed to support atmospheric modeling is Atmol [52]. Atmol provided a DSL to allow scientists to describe their models using partial differential equations operators. Later, Liszt [14] provided a DSL for constructing mesh-based PDE solvers targeting heterogeneous systems.

Multi-target support was also provided by Physis [42]. Physis is a C-based DSL which allows developing models using structured grids.

Another form of DSL is Icon DSL [50]. Icon DSL was developed to apply index interchanges based on described swapping on Fortran-based models.

Further work based on C++ constructs and generic programming to improve performance portability is Stella [23] and later GridTools [13]. Computations are specified with a C++-based DSL and the tools generate code for CPUs or GPUs. GridTools are used to port some kernels from the NICAM model in our project AIMES to evaluate existing DSLs.

Although C++ provides strong features through generic programming allowing to avoid performance portability issues, scientists are reluctant to utilize alternative programming languages as the existing codes are huge. Normally scientists prefer to keep using preferred languages, e.g. Fortran, rather than moving to learning C++ features.

Other forms of DSLs used directives to drive code porting or optimization. Hybrid Fortran [43], HMPP [16], Mint [51], CLAW [12] are examples of directive-based approach. Such solutions allow adding directives to code to guide some optimization. Scientists write code in some form and add directives that allow tools to provide specific features, e.g., CLAW allows writing code for one column and allows using directives to apply simulations over the set of columns in parallel.

In the MetOffice's LFRic model [1], a DSL is embedded into the Fortran code that provides an abstraction level suitable for the model. The model ships with the PSyclone code-generator that is able to transform the code for different target platforms. In contrast to our lightweight solution, these DSLs are statically defined and require a big translation layer.

2.2 Compression

Data-reduction techniques related to our work can be structured into: (1) algorithms for the lossless data compression; (2) lossy algorithms designed for scientific (floating-point) data and the HPC environment; (3) methods to identify necessary data precision and for large-scale evaluation.

Lossless Algorithms There exist various lossless algorithms and tools, for example, the LZ77 [55] algorithm which utilizes dictionaries. By using sliding windows, it scans uncompressed data for two largest windows containing the same data and replaces the second occurrence with a pointer to the compressed data of the first. The different lossless algorithms vary in their performance characteristics and ability to compress data depending on its characteristics. A key limitation is that users have to pick an algorithm depending on the use case. In [25], we presented compression results for the analysis of typical climate data. Within that work, the lossless compression scheme MAFISC with preconditioners was introduced. With MAFISC, we also explored the automatic selection of algorithms by compressing each block with two algorithms, the best compression chain so far and one randomly chosen. It compresses data 10% more than the second best algorithm (e.g., standard compression tools).

Lossy Algorithms for Floating-Point Data SZ [15] and ZFP [41] are the de-facto standard for compressing floating-point data in lossy mode. Both provide a way of bounding the error (either bit precision or absolute error quantities) but only one quantity can be selected at a time. ZFP [41] can be applied up to three dimensions. SZ is a newer and effective HPC data compression method; it uses a predictor and the lossless compression algorithm GZIP. Its compression ratio is typically significantly better than the second-best solution of ZFP. In [26], two lossy compression algorithms (GRIB2, APAX) were evaluated regarding the loss of data precision, compression ratio, and processing time on synthetic and climate dataset. These two algorithms have equivalent compression ratios and depending on the dataset APAX signal quality may exceed GRIB2 and vice versa.

Methods The application of lossy techniques to scientific (floating-point) datasets is discussed in [11, 22, 27, 38–40]. A statistical method to predict characteristics (such as proportions of file types and compression ratio) of stored data based on representative samples was introduced in [34] and the corresponding tool in [35]. It can be used to determine compression ratio by scanning a fraction of the data, thus reducing costs.

Efforts for determination of appropriate levels of precision for lossy compression methods for climate and weather data were presented in [3] and in [4]. The basic idea is to compare the statistics derived from the data before and after applying lossy compression schemes; if the scientific conclusions drawn from the data are similar and indistinguishable without the compression, the loss of precision is acceptable.

www.dbooks.org

3 Towards Higher-Level Code Design

Computations in earth system modeling run hundreds or thousands of stencil computations over wide grids with huge numbers of points. Such computations are time consuming and are sensitive to optimal use of computer resources. Compilers usually apply a set of optimizations while compiling code. Semantical rules of the general purpose language (GPL) can be applied to the source code and used within compilers to translate the code to semantically equivalent code that runs more efficiently. However, often the semantical information extracted from the source code are not enough to apply all relevant optimizations as some high-level optimizations would alter the semantics—and the rules of the GPL forbid such changes. As mostly code from GPLs is at a lower semantical level than the abstraction level of the developers, opportunity of optimization is lost. Such lost opportunities are a main obstacle to develop software in a performance-portable way in earth system modeling.

To address the lost opportunities of optimization, different techniques are applied by the scientists directly to the source code. Thus, it is the responsibility of the scientists who develop the models to write the code with those decisions and guidelines on optimizations in mind. Drawbacks of this strategy include pushing scientists to focus on machine details and optimal code design to use hardware resources. Scientists need to learn the relevant optimization techniques from computer science such as, e.g., cache blocking.

3.1 Our Approach

To solve the issues with the manual code optimization, we suggest using an additional set of language constructs which exhibit higher-level semantics. This way, tools can be used to apply optimizations based on those semantics. Optimization responsibilities are moved again from scientists to tools.

As usually, the source code is developed by scientists, however, in our approach, instead of coding on low-level and caring for optimization strategies, a DSL is used as abstraction. Machine-specific or computer scientific concepts are not needed to write the source code of a model. This is enabled by increasing the abstraction level of a GPL by providing a language extension with semantics based on the scientific concepts from the domain science itself.

The DSL implements a template mechanism to simplify and abstract the code. For the purpose of this project, it was designed to abstract climate and weather scientific concepts but other domains and models could be supported. Therefore, stencils and grids are used as the basis in the DSL. The language extensions hide the memory access and array notations. They also hide the details of applying the stencils and the traversal of the grids. The grid structure itself is hidden in the source code. A model's code uses the grid without specifying the locations of the grid

points in memory or how neighbors are accessed. All such details are specified in the configuration files. Lower details are neglected at the DSL level.

Translation tools handle the processing of the higher-level source code and converting it to compatible GPL code. The semantics of the added language extensions are extracted from the source code and are used to apply further high-level transformations to the code. Applied transformations are guided by configuration files that allow users to control the optimization process and may convert the code to various back-end representations that, in-turn, can be converted to code that is executed on a machine. A key benefit of this strategy is that it increases the performance-portability: A configuration can apply optimization techniques exploiting features of a specific architecture. Therefore, a single scientific code base can be used to target multiple different machines with different architectures.

Model-specific configuration files are provided separately to guide the code translation and optimization process. Those files are developed by scientific programmers rather than domain scientists. In contrast to domain scientists, the scientific programmers must have an intermediate understanding of the scientific domain but also understand the hardware architecture of a target machine. They use their experience to generate code that uses the machine's resources, and write the configurations that serve the purpose of optimal use of that specific machine to run the selected model.

Our approach offers separation of concerns between the parties. Scientific work is done by scientists and optimization is done by scientific programmers. The concept of the approach is illustrated in Fig. 2. For the icosahedral models, a single intermediate domain language is derived that can be adjusted for the needs of each model individually (dialects). From this single source various code representations (back-ends) could be generated according to configuration, e.g., MPI+OpenMP or GASPI.

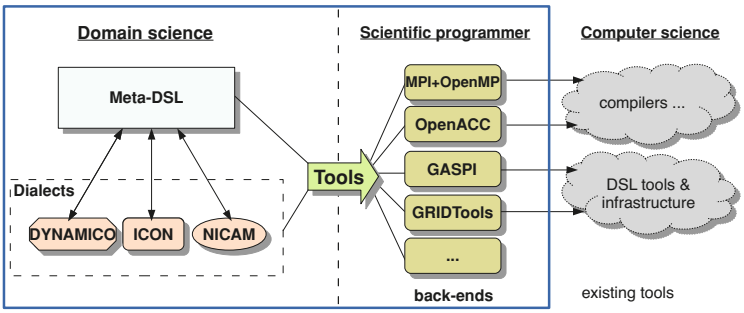


Fig. 2 Separation of concerns

www.dbooks.org

3.2 Extending Modeling Language

An important point we consider in our approach is keeping the general-purpose language, e.g. Fortran, that the scientists prefer to use to develop their model. We add additional language constructs to the GPL. This simplifies the mission to port existing models which could include hundreds of thousands of lines of code. The majority of the code is kept, while porting some parts incrementally; by providing templates in the configuration files, code can be simplified while it still produces equivalent GPL constructs: Changes are replacements of loop details and field access into alternative form using the added extensions. A drawback of the incremental approach is that the full beauty of optimizations like memory adjustments requires to have ported the complete model.

Our plan to develop such language extensions was to start with the three existing modern icosahedral models of the project partners. In the first phase, special dialects were proposed to support each model. Then, we identified common concepts and defined a set of language extensions that support the domain with domain-specific language constructs. We collected requirements, and worked in collaboration with scientists from the three models to reach at the language extensions, in detail:

1. The domain scientists suggested compute-intensive and time-consuming code parts.
2. We analyzed the chosen code parts to find out possibilities to use scientific terms instead of existing code. We always kept in mind that finding a common representation across the three models leads to domain-specific language extensions.
3. We replaced codes with suggested extensions.
4. We discussed the suggestions with the scientists. Discussions and improvements were done iteratively. The result of those discussions lead to the GGDML language extensions.

3.2.1 Extensions and Domain-Specific Concepts

The *General Grid Definition and Manipulation Language (GGDML)* language extensions provide a set of constructs for:

- Grid definition
- Declaration of fields over grids
- Field access and update
- Grid traversal to apply stencils
- Stencil reduction expressions

GGDML code provides an abstraction including the order of the computation of elementary operators. Therefore, optimizations can result in minor changes of the computed result of floating-point operations; this is intentional as bit-reproducibility constraints the optimization potential.

In code that is written with GGDML, scientists can specify the name of the grid that should be traversed when applying a stencil. The definitions of the grids are provided globally through the configuration files. GGDML allows to specify a modified set of grid points rather than the whole set of grid points as provided through the grid definition grid, e.g., traversing a specific vertical level. Such possibilities are offered through naming a grid and using operators that allow adding, dropping, or modifying dimensions of that grid. Operators could change the dimensionality of the grid or override the existing dimensions.

Fields are declared over different grids through declaration specifiers. GGDML provides a flexible solution to support application requirements. A basic set of declaration specifiers allows to control the dimensionality of the grid, and the localization of the field with respect to the grid. Such declaration specifiers allow applications to deal with surfaces and spaces, and also supports using staggered as well as collocated grids.

Access specifiers provide tools the necessary information that will be used to allocate/deallocate and access the fields. Field access is an important part of stencil operations. GGDML provides an iterator statement to apply the stencil operations over a set of grid points. The GGDML iterator statement replaces loops and the necessary optimization to apply stencils. It provides the user an index that refers to the current grid point. Using this index, scientists can write their stencils without the need to deal with the actual data structures that hold the field data. The iterator applies the body to each grid point that is specified in the grid expression, which is one part of the iterator statement. This expression is composed from the name of a grid, and possibly a set of modifications using operators as mentioned above.

The iterator's index alone is not sufficient to write stencil operations, as stencils include access to neighboring points. For this purpose, GGDML uses access operators, which represent the spatial relationships between the grid points. This allows to access the fields that need to be read or written within a stencil operation using spatial terms instead of arrays and memory addresses. To support different kinds of grids, GGDML allows users to define those access operators according to the application needs.

Repetitions of the same mathematical expressions over different neighbors is common in stencil operations. To simplify writing stencils, GGDML provides a reduction expression. Reduction expressions apply a given sub-expression over multiple neighbors along with a mathematical operator applied to the set of the subexpressions.

3.3 Code Example

To demonstrate the code written with extensions, a sample code from the NICAM model written with Fortran is given in Listing 1. As we can see from the original NICAM code, a pattern is repeated in the code: the same field is accessed multiple times over multiple indices. Optimization is limited as firstly, the memory layout

www.dbooks.org

is hardcoded in the fields cgrad and scl, secondly the iteration order is fixed. Integrating blocking for cache optimization in this schema would increase the complexity further.

Listing 1 NICAM Fortran code

```
do d = 1, ADM_nxyz
  do l = 1, ADM_lall
!OCL PARALLEL
    ! support indices to address neighbors
    do k = 1, ADM_kall
       do n = OPRT_nstart, OPRT_nend
          ij     = n
          ip1j   = n + 1
          ijp1   = n     + ADM_gall_1d
          ip1jp1 = n + 1 + ADM_gall_1d
          im1j   = n - 1
          ijm1   = n     - ADM_gall_1d
          im1jm1 = n - 1 - ADM_gall_1d

          grad(n,k,l,d) = cgrad(n,1,0,d) * scl(ij    ,k,l) &
                        + cgrad(n,1,1,d) * scl(ip1j  ,k,l) &
                        + cgrad(n,1,2,d) * scl(ip1jp1,k,l) &
                        + cgrad(n,1,3,d) * scl(ijp1  ,k,l) &
                        + cgrad(n,1,4,d) * scl(im1j  ,k,l) &
                        + cgrad(n,1,5,d) * scl(im1jm1,k,l) &
                        + cgrad(n,1,6,d) * scl(ijm1  ,k,l)
       enddo
       grad(          1:OPRT_nstart-1,k,l,d) = 0.0_RP
       grad(OPRT_nend+1:ADM_gall     ,k,l,d) = 0.0_RP
    enddo
  enddo
enddo
```

The same semantics rewritten with the DSL is shown in Listing 2. Instead of iterating across the grid explicitly, a FOREACH loop specifies to run on each element of the grid, the coordinates are encoded in the new cell variable. We reduced the repeated occurrences of the fields with the indices with a 'REDUCE' expression. The Fortran indices are replaced with DSL indices that made it possible to simplify the field access expressions.

Listing 2 NICAM DSL code

```
FOREACH cell in grid | g{OPRT_nstart..OPRT_nend}
  do d = 1, ADM_nxyz
       grad(cell,d) = REDUCE(+,N={0..6},
           cgrad(cell%g,cell%l,N,d) * scl(cell%neighbor(N))
               ↪ )
  enddo
END FOREACH
```

```
FOREACH cell in GRID%cells | g{1..OPRT_nstart-1 , OPRT_nend+1
    ↪  .. gall}
  do d = 1, ADM_nxyz
           grad(cell,d) = 0.0_PRECISION
  enddo
END FOREACH
```

3.4 *Workflow and Tool Design*

Model code that is based on the DSL along with code of the model in general-purpose language goes through a source-to-source translation process. This step is essential to make the higher-level code ready for processing by the compilers. Our tool is designed in a modular architecture. By applying a configuration, it translates code in a file or a source tree and generates a version of the transformed code. The generated code is the optimized version for a specific machine/architecture according to the configuration. Inter-file optimizations in code trees can also be detected and applied.

The tool is implemented with Python3. Users call the main module with parameters that allow them to control the translation process, e.g. to specify a language module. The code tree is provided to the main module also as an argument.

The main module loads the other necessary modules. The DSL handler module constructs the necessary data structures according to the user-provided configuration file. The source code tree is then parsed into abstract syntax trees (AST). The generated ASTs can be analyzed for optimization among the source files. After all the optimizations/transformations are applied, the resulting code tree is serialized to the output file. Figure 3 shows the design of the translation process.

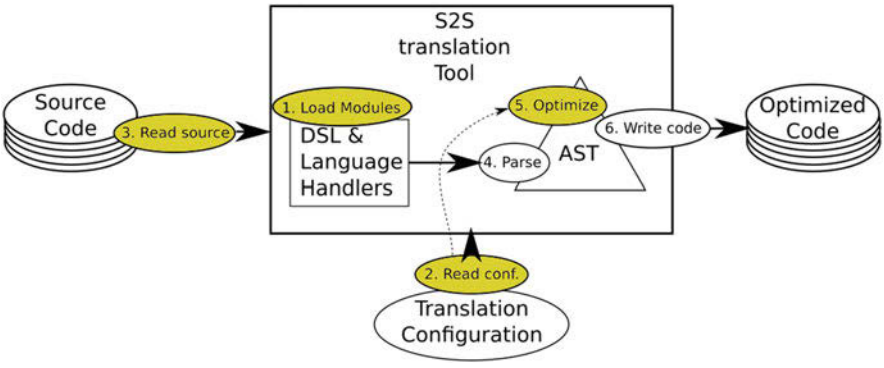


Fig. 3 Translation process. Yellow components are influenced by the user options

www.dbooks.org

3.5 Configuration

Configuration files include multiple sections, among which some are essential and others can be added only if needed. Optimization procedures are driven by those configuration sections. The translation tool uses defaults in case optional sections are missing.

Blocking Among the important optimizations that help improve the performance of stencil computations is the optimal use of the caches and memory access. Reusing the data in the caches eliminates the need to read the same data elements repeatedly from memory. Often, data locality can be exploited in stencil computations, allowing for performance improvements.

Cache blocking is a technique to improve the data reuse in caches. Our translation process can apply cache blocking based on the scientific programmer's recommendations. One configuration section is used to allow to specify cache blocking information. The default when this section is missing in a configuration file is to not apply cache blocking.

An example kernel using GGDML in the C programming language is shown in Listing 3.

Listing 3 Example kernel using C with GGDML

```
foreach c in grid
{
  float df=(f_F[c.east_edge()]-f_F[c.west_edge()])/dx;
  float dg=(f_G[c.north_edge()]-f_G[c.south_edge()])/dy;
  f_HT[c]=df+dg;
}
```

Applying cache blocking using a configuration file defining 10,000 elements per block generates the loop code shown in Listing 4. The first loop handles completely occupied blocks with 10 k elements and the second loop the remainder. In both cases, the loop body (not shown) contains the generated stencil code.

Listing 4 Example loop structure for blocking

```
for (size_t blk_start = (0); blk_start < (GRIDX); blk_start
    ↪ +=10000){
    size_t blk_end = GRIDX;
    if ((blk_end - blk_start) > 10000)
        blk_end = blk_start + 10000;
    // Generated loop body
}
#pragma omp simd
for (size_t XD_index = blk_start; XD_index < blk_end;
    ↪ XD_index++) {
  // Generated loop body
}
```

Memory Layout Another important point to optimize memory bandwidth is to optimize the data layout in memory. The temporal and spacial locality of data should lead to access of data in complete cache lines such that it can be prefetched and cached effectively. Thus, data that is accessed in short time frames should be stored closer in memory. To exploit such possibilities, our translation tool provides a flexible layout transformation procedure. The DSL itself abstracts from data placement, however, the translation process generates the actual data accesses. This layout specification is described in configuration files.

Besides to data layout control, the loop nests that access the field data are also subject to user control. The order of the loops that form a nested loop is critical for the optimal data access. Loop order of loops that apply the stencils is also controlled by configuration files.

Listing 5 illustrates the resulting code after using a data layout transformation. In this case, a 2D grid is stored in a single-dimensional array.

Listing 5 Example code generated with index transformation

```
[...]
#pragma omp for
for (size_t YD_index = (0); YD_index < (local_Y_Cregion);
    ↪ YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
       ↪ XD_index++){
    float df = (f_F[(YD_index + 1) * (GRIDX + 3) + (XD_index
        ↪ + 1)+1]
        - f_F[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1])
            ↪  * invdx;
    float dg = (f_G[((YD_index + 1)+1) * (GRIDX + 3) + (
        ↪ XD_index)+1]
        - f_G[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1])
            ↪  * invdy;
    f_HT[(YD_index + 1) * (GRIDX + 3) + (XD_index) + 1] = df
        ↪ + dg;
   }
}
[...]
```

Inter-Kernel Optimization Cache blocking and memory layout allow improving the use of the caches and memory bandwidth at the level of the kernel. However, the optimal code at the kernel level does not yet guarantee optimal use of caches and memory bandwidth at the application level. Consider the example where two kernels share most of their input data but compute different outputs independently from each other. These kernels could be fused together and benefit from reusing cached data. Note that the benefit is system specific, as the size of the cache and application kernel determine the optimal size for blocking and fusion.

www.dbooks.org

The inter-kernel optimization allows exploiting such data reuse across kernels. To exploit such potential, our translation tool can run an optimizer procedure to detect such opportunities and to apply them according to user decision of whether to apply any of those optimizations or not.

Therefore, the tool analyzes the calls among the code files within the code tree. This analysis leads to a list of call inlining possibilities. The inlining could lead to further optimization through loop fusions. The tool runs automatic analysis including data dependency and code consistency. This analysis detects possible loop fusions that still keep computations consistent. Such loop fusion may lead to optimized use of memory bandwidth and caches. We tested the technique experimentally (refer to Sect. 4.5) to merge kernels in the application. We could improve the use of caches and hence the performance of the application with 30–48% on different architectures.

The tool provides a list of possible inlining and loop fusion cases. Users choose from the list which case to apply—we anticipate that scientific programmers will make an informed choice for a target platform based on performance analysis tools. According to the choice that the user makes, the tool applies the corresponding transformations automatically.

Listing 6 shows two kernels to compute the two components of the flux.

Listing 6 Example code with two kernels to compute flux components

```
[...]
#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Eregion);
    ↪ YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
      XD_index++) {
      f_F[YD_index][XD_index] = f_U[YD_index][XD_index] *
      (f_H[YD_index][XD_index] +f_H[YD_index][XD_index -1])
          ↪ /2.0;
   }
}
[...]
#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Eregion);
    ↪ YD_index++){
#pragma omp simd
   for (size_t XD_index = blk_start; XD_index < blk_end;
       ↪ XD_index++){
      f_G[YD_index][XD_index] = f_V[YD_index][XD_index] *
      (f_H[YD_index][(XD_index] + f_H[YD_index -1][XD_index])
          ↪ /2.0;
   }
}
[...]
```

Listing 7 shows the resulting code when the two kernels are merged.

Listing 7 Merged version of the flux computation kernels

```
[...]
#pragma omp parallel for
for (size_t YD_index = 0; YD_index < (local_Y_Eregion);
    ↪ YD_index++){
#pragma omp simd
  for (size_t XD_index = blk_start; XD_index < blk_end;
      ↪ XD_index++) {
    f_F[YD_index][XD_index] = f_U[YD_index][XD_index] *
    (f_H[YD_index][XD_index] + f_H[YD_index][(XD_index) +
        ↪ (-1)])/2.0;

    f_G[YD_index][XD_index] = f_V[YD_index][XD_index] *
    (f_H[YD_index][(XD_index] + f_H[(YD_index) + (-1)][
        ↪ XD_index])/2.0;
  }
}
[...]
```

Utilizing Distributed Memory Beyond parallelization on the node resources, our techniques allow scaling the same source code that uses GGDML over multiple nodes utilizing distributed memory. This is essential to run modern models on modern supercomputer machines.

The GGDML code is unaware of underlying hardware, and does not need to be modified to run on multiple nodes. Rather, configuration files are prepared to translate the GGDML code into code that is ready to be run on multiple nodes. Configuration files allow domain decomposition to distribute the data and the computation over the nodes. Necessary communication of halo regions is also enabled through configuration files. Scientific programmers can generate simple parallelization schemes, e.g., MPI using blocking communication or sophisticated alternatives like non-blocking communication. When using non-blocking communication, a further optimization is to decouple the computation of the inner region that can be calculated without needing updated halo data and outer regions that require data from another process to be computed.

The translation tool extracts neighborhood information from the GGDML extensions. Such extracted information is analyzed by the tool to decide which halo regions should be exchanged between which nodes. Decisions and information from configuration files allow to generate the necessary code that handles the communication and the synchronization. This guarantees that the necessary data are consistent on the node where the computation takes place.

The parallelization is a flexible technique. No single library is used to handle the parallelization, rather, the communication library is provided through configuration files. Thus, different libraries or library versions can be used for this purpose. We have examined the use of MPI and GASPI as libraries for parallelization.

www.dbooks.org

Listing 8 shows the resulting communication code of halo regions between multiple processes—in this case without decoupling of inner and outer area, code with decoupled areas is longer. In this listing, dirty flags are generated to communicate only necessary data. Flags are set global to all processes, and can be checked by each process such that processes that need to do communication can make use of them. This way we guarantee to handle communication properly.

Listing 8 Example generated code to handle communication of halo data

```
[...]
//part of the halo exchange code
if (f_G_dirty_flag[11] == 1) {
  if (mpi_world_size > 1) {
    comm_tag++;
    int pp = mpi_rank != 0 ? mpi_rank - 1 : mpi_world_size -
        ↪ 1;
    int np = mpi_rank != mpi_world_size - 1 ? mpi_rank + 1 :
        ↪ 0;
    MPI_Isend(f_G[0], GRIDX + 1, MPI_FLOAT, pp,
              comm_tag, MPI_COMM_WORLD, &mpi_requests[0]);
    MPI_Irecv(f_G[local_Y_Eregion], GRIDX + 1, MPI_FLOAT, np,
              comm_tag, MPI_COMM_WORLD, &mpi_requests[1]);
    MPI_Waitall(2, mpi_requests, MPI_STATUSES_IGNORE);
[...]

#pragma omp parallel for
for(size_t YD_index = (0); YD_index < (local_Y_Cregion);
    ↪ YD_index++){
#pragma omp simd
  for (size_t XD_index = blk_start; XD_index < blk_end;
      ↪ XD_index++){
   float df = (f_F[YD_index][(XD_index) + (1)]
      - f_F[YD_index][XD_index]) * invdx;
   float dg = (f_G[(YD_index) + (1)][XD_index]
      - f_G[YD_index][XD_index]) * invdy;
   f_HT[YD_index][XD_index] = df + dg;
  }
}
[...]
```

3.6 *Estimating DSL Impact on Code Quality and Development Costs*

To estimate the impact of using the DSL on the quality of the code and the costs of model development, we took two relevant kernels from each of the three icosahedral models, and analyzed the achieved code reduction in terms of lines

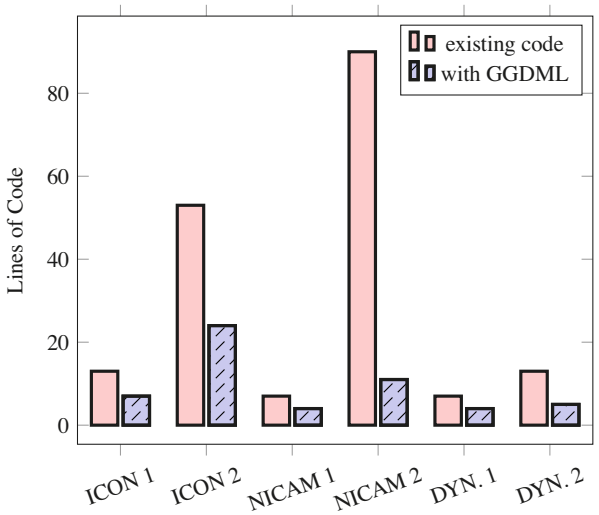


Fig. 4 GGDML impact on the LOC on several scientific kernels [32]

of code (LOC) [32]. We rewrote the kernels (originally written in Fortran) using GGDML + Fortran. Results are shown in Fig. 4.

The average reduction in terms of LOC is 70%, i.e. LOC in GGDML+Fortran in comparison to original Fortran code is 30%. More reduction is noticed in some stencils (NICAM example No.2, reduced to 12%).

Influence on readability and maintainability: Using COCOMO [6] as a model to estimate complexity of development effort and costs, we estimated in Table 1 the benefits as a result of the code reductions when applying GGDML to develop a model comparable to the ICON model. The table shows the effort in person month, development time and average number of people (rounded) for three development modes: the embedded model is typically for large project teams a big and complex code base, the organic model for small code and the semi-detached mode for in-between. We assume the semi-detached model is appropriate but as COCOMO was developed for industry projects, we don't want to restrict the development model. The estimations are based on a code with 400KLOC, where 300KLOC of the code are the scientific portion that allows for code reduction while 100KLOC are infrastructure.

From the predicted developed effort, it is apparent that the code reductions would be leading to a significant effort and cost reduction that would justify the development and investment in DSL concepts and tools.

4 Evaluating Performance of our DSL

To illustrate the performance benefits of using the DSL for modeling, we present some performance measurements that were measured for example codes written with the DSL and translated considering different optimization aspects (configura-

www.dbooks.org

Table 1 COCOMO cost estimates [32]

Development Style	Codebase	Effort applied (person month)	Dev. Time (months)	People required	Dev. costs (M€)
Embedded	Fortran	4773	37.6	127	23.9
	DSL	1133	28.8	72	10.4
Semi-detached	Fortran	2462	38.5	64	12.3
	DSL	1133	29.3	39	5.7
Organic	Fortran	1295	38.1	34	6.5
	DSL	625	28.9	22	3.1

tions). Two different testcodes were used to evaluate the DSL's support for different types of grids: One application uses an unstructured triangular grid, and the other uses a structured rectangular grid that could be applied for code in cubic sphere. Both were written using GGDML (our DSL) in addition to C as the host modeling language.

4.1 Test Applications

Laplacian Solver The first application code uses an unstructured triangular grid covering the surface of the globe. The application was used in the experiments to apply the Laplacian operator of a field at the cell centers based on field values at neighboring cells. Generally, this code includes fields that are localized at the cell centers, and on the edges of the cells. The horizontal grid of the globe surface is mapped to a one dimensional array using Hilbert space-filling-curve. We used 1,048,576 grid points (and more points over multiple-node runs) to discretize the surface of the globe. The code is written with 64 vertical levels. The surface is divided into blocks. The kernels are organized into components, each of which resembles a scientific process.

Shallow Water Model The other code is a shallow water equation solver. It is developed with a structured grid. Structured grids are also important to study for icosahedral modeling, as some icosahedral grids can be structured. Fields are located at centers of cells and on edges between cells. This solver uses the finite difference method. The test code is available online.[1] As part of the testing, we investigate performance portability of code developed using the DSL.

4.2 Test Systems

The experiments were executed in different times during the course of the project and used different machines based on availability and architectural features.

- Mistral
  The German Climate Computing Center provides nodes with Intel(R) Xeon(R) E5-2695 v4 (Broadwell) @ 2.1 GHz processors.
- Piz Daint
  The Swiss supercomputer provides nodes equipped with two Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60 GHz processors and NVIDIA(R) Tesla(R) P100 GPUs.

[1] https://github.com/aimes-project/ShallowWaterEquations.

www.dbooks.org

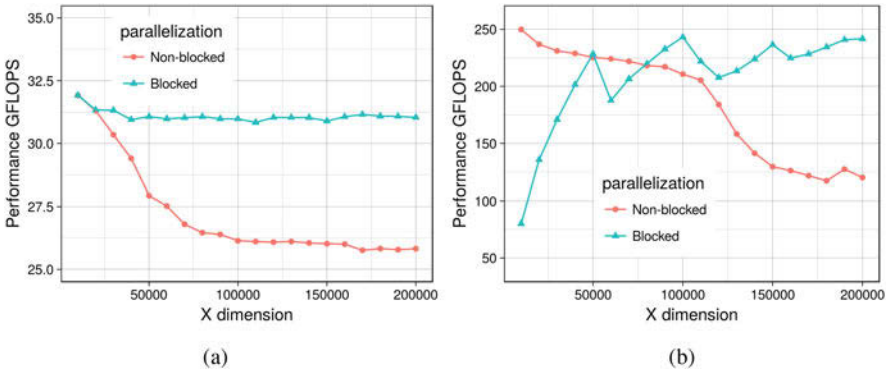


Fig. 5 Impact of blocking: Performance measurements with variable grid width. (**a**) Broadwell CPU. (**b**) P100 GPU

- NEC test system
  Intel(R) Xeon(R) CPU E5-2697 v4 @ 2.30 GHz Broadwell processors with Aurora vector engines.

4.3 Evaluating Blocking

To explore the benefit of organizing memory accesses efficiently across architectures, experiments using the shallow water equation solver code were conducted on Piz Daint. First, we generated code versions with and without blocking for the Broadwell processor and the P100 GPU. An excerpt of results presented for different size of grids is shown in [29] and in Fig. 5a. The experiment was done with grid width of 200K. In the paper, we investigated the influence of the blocking factor on the application kernel further revealing that modest block sizes are leading to best performance (256 to 10 k for CPU and 2–10 k for GPU). On both architectures, wider grids run less efficiently as a result of an inefficient use of caches. The GPU implies additional overhead for the blocked version, requiring to run with a sufficiently large grid to benefit from it. This also shows the need to dynamically turn on/off blocking factors depending on the system capabilities.

4.4 Evaluating Vectorization and Memory Layout Optimization

As part of [28], we evaluated techniques to apply memory layout transformations. The experiments were done on two architectures, the Broadwell multi-core processors and the Aurora vector engine on the NEC test platform using the shallow water equation solver code.

Table 2 Performance of memory layout variants on CPU and the NEC vector architecture

Architecture	Scattered	Short distance	Contiguous
Broadwell	3 GFlops	13 GFlops	25 GFlops
NEC Aurora	80 GFlops	161 GFlops	322 GFlops

We used alternative layouts with different distances between data elements to investigate the impact of the data layout on the performance. The explored data alternatives were data accesses to

- contiguous unit stride arrays,
- interleaved data with constant short distance separating data elements, 4 bytes separating consecutive elements. This allowed to emulate array of structures (AoS) layouts,
- scattered data elements separated with long distances.

The results are listed in brief in Table 2. Using memory profilers, we found that the contiguous unit-stride code allowed to use the memory throughput efficiently on both architectures. In the emulated AoS codes, the efficiency dropped on both architectures. The worst measurements were taken for the scattered data elements.

Besides the impact of the optimization on the use of the memory bandwidth, vectorization is also affected by those alternatives. AVX2 was used for all kernels on Broadwell for the unit-stride code. Similarly, the vector units of the vector engine showed best results with this layout. Again the use of the vectorization was degraded with the emulated AoS, and was even worse with the scattered data elements.

4.5 Evaluating Inter-Kernel Optimization

To explore the benefit of kernel merging, experiments were done using the shallow water equation solver code on the NEC test system (for vector engines) and Piz Daint (for GPUs and CPUs) [29]. The performance of regular and merged kernels are shown in Table 3. The code achieves near optimal use of the memory bandwidth already before the merge and actually decreases slightly after the merge. Exploiting the inter-kernel cache reuse allowed to reduce the data access in memory and increased the total performance of the application by 30–50%.

4.6 Scaling with Multiple-Node Runs

To demonstrate the ability of the DSL to support global icosahedral models, we carried out experiments using the two applications. Scalability experiments of

www.dbooks.org

Table 3 Performance and efficiency of the kernel fusioning on all three architectures

Architecture	Theoretical Memory bandwidth (GB/s)	Before merge		With Inter-Kernel Merging	
		Measured memory throughput (GB/s) and peak	GFlops	Measured memory throughput (GB/s) and peak	GFflops
Broadwell	77	62 (80%)	24	60 (78%)	31 (+30%)
P100 GPU	500	380 (76%)	149	389 (78%)	221 (+48%)
NEC Aurora	1,200	961 (80%)	322	911 (76%)	453 (+40%)

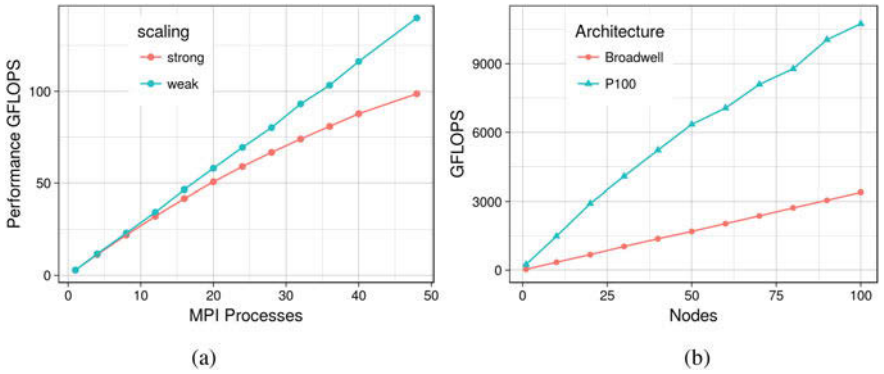


Fig. 6 Scalability of the two different models (measured on Mistral; P100 on Piz Daint). (**a**) Icosahedral code [30]. (**b**) Shallow water solver [31]

unstructured grid code were run on Mistral. Shallow water equation solver code experiments were run on Mistral for CPU tests, and on Piz Daint for GPU tests.

In the experiment using the unstructured grid, we use the global grid of the application and apply a three-dimensional Laplacian stencil. We varied the number of nodes that we use to run the code up to 48 nodes. The minimum number of the grid points we used is 1,048,576. We used this number of points for the strong-scale analysis. Weak scalability experiments were based on this number of points for each node. Figure 6a shows the results.

We could do further numbers of nodes, however, we found that the code was scaling with the tested cases and further experiments needed resources and time to get jobs to run on the test machine. For the measured cases, the weak scalability of the code is close to optimal. Thus, increasing the resolution of the grids and running the code on more nodes is achieved efficiently. This is an important point as higher resolution grids are essential for recent and future global simulations.

We also carried out experiments to scale the shallow water equation solver on both Broadwell multi-core processors and on the P100 GPUs at Piz Daint. We generated code for Broadwell experiments with OpenMP as well as MPI, and for the GPUs with OpenACC as well as MPI. Figure 6b shows the measured results of scaling the code. On the Broadwell processor, we used 36 OpenMP threads on each node.

While the performance of the GPU code scaled well, it loses quite some efficiency when running on two processes as the halo communication involves the host code. In general, the code that the tools generate for multiple nodes shows to be scalable, both on CPUs and GPUs. The DSL code does not include any details regarding single or multiple node configuration, so users do not need to care about multiple node parallelization. However, the parallelization can still be applied by the tool and the users can still control the parallelization process.

4.7 Exploring the Use of Alternative DSLs: GridTools

The GridTools framework is a set of libraries and utilities to develop performance-portable weather and climate applications. It is developed at The Swiss National Supercomputing Center [13]. To achieve the goal of performance portability, the user code is written in a generic form which is then optimized for a given architecture at compile time. The core of GridTools is the stencil composition module which implements a DSL embedded in C++ for stencils and stencil-like patterns. Further, GridTools provides modules for halo exchanges, boundary conditions, data management and bindings to C and Fortran. GridTools is successfully used to accelerate the dynamical core of the COSMO model with improved performance on CUDA-GPUs compared to the current official version, demonstrating production quality and feature-completeness of the library for models on lat-lon grids [48]. Although GridTools was developed for weather and climate applications it might be applicable for other domains with a focus on stencil-like computations.

In the context of the AIMES project, we evaluated the viability of using GridTools for the dynamical core of NICAM: namely NICAM-DC. Since NICAM-DC is written in Fortan, we first had to port the code to C++, which includes also changing the build systems. Figures 9 and 10 show simple example codes extracted from NICAM-DC and ported to GridTools, respectively. We ported the dynamical core using the following incremental approach. First, each operator was ported individually to GridTools, i.e. re-written from Fortran to C++. Next, we used a verification tool to assure that the same input to the C++ and Fortran version gives the same output. Next we move on to the following operator. Table 4 shows results from benchmarks extracted from NICAM-DC. It provides good speedup on GPU, and speed on CPU (in OpenMP) comparable to the hand-written version. Figure 7 shows results for running all operators of NICAM-DC on 10 nodes. It is worth mentioning that the most time consuming operator is more than 7x faster on GPU versus CPU.

GridTools demonstrates good GPU performance and acceptable CPU performance. The functionalities and features included in GridTools were enough to support the regular mesh code of NICAM-DC without friction (i.e. no custom features were required in GridTools to support the requirements of NICAM-DC). In addition, GridTools are transparent in the sense that no information about the platform is exposed to the end-user. On the other hand, following is a list of issues

www.dbooks.org

Table 4 Execution time (seconds) of different benchmarks extracted from NICAM-DC. This includes the regular regions only, using 1 region, a single MPI rank for a 130 × 130 × 42 grid

Benchmark	CUDA (Nvidia K40)			OpenMP (Broadwell-EP CPU E5-2630 v4 @2.20GHz)					
	Manual	Manual_opt (coal, sh_mem, occu, reg_pres)	GridTools	OMP_THREADS=1		OMP_THREADS=5		OMP_THREADS=10	
				Manual	GridTools	Manual	GridTools	Manual	GridTools
Diffusion	5.83	0.575 (5.23x OMP=10)	0.61 (4.93)	19.2 (1.0x)	19.4 (1.0x)	4.3 (4.46x)	5.8 (3.34x)	2.2 (8.72x)	3.01 (3.22x)
Divdamp (vgrid40_600m_24km)	35.23	3.15 (4.83x OMP=10)	3.17 (4.80x)	74.3 (1.0x)	74.3 (1.0x)	15.4 (4.81x)	30.08 (2.47x)	7.7 (4.78x)	15.22 (2.44x)
Vi_rhow_Solver (vgrid40_600m_24km)	0.91	0.288 (6.14x OMP=10)	0.311 (5.69x)	12.0 (1.0x)	12.1 (1.0x)	2.4 (4.9)	3.18 (3.8x)	1.23 (4.87)	1.77 (3.4x)

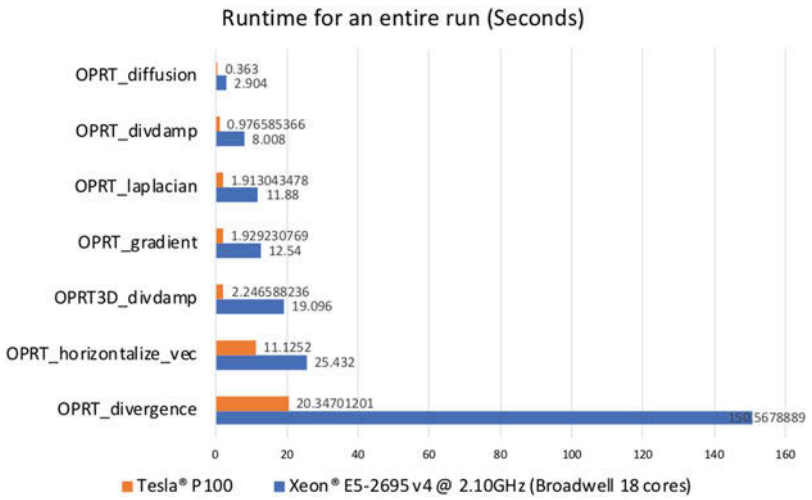


Fig. 7 NICAM-DC operators. Running on 10 nodes with one MPI rank per node. P100 is running GridTools generated kernels and Xeon uses the original Fortran code. Total grid is 32 × 32 × 10 using 40 vertical layers

that requires one's consideration when using GridTools: first, the requirement of rewriting the entire dynamical core in C++ is not a trivial task, especially since C++ templates make the code more convoluted, in comparison to Fortran. Second, while GridTools as a stencil framework does a good job for the dynamical core, separate solutions are required for the physics modules, the communicator module, and the non-regular compute modules (e.g. polar regions). Using different solutions inside the same code base typically increases the friction between code modules. Third, the interfacing between Fortran and C++ is non-trivial and can be troublesome considering that not all end-users are willing to change their build process.

Listing 9 Example of the diffusion operator extracted from NICAM-DC

```
do d = XDIR, ZDIR
do j = jmin-1, jmax
do i = imin-1, imax
  vt(i,j,d) = (( + 2.0_RP * coef(i,j,d,1) &
                 - 1.0_RP * coef(i,j,d,2) &
                 - 1.0_RP * coef(i,j,d,3) ) * scl(i,j,d) &
             +  ( - 1.0_RP * coef(i,j,d,1) &
                  + 2.0_RP * coef(i,j,d,2) &
                  - 1.0_RP * coef(i,j,d,3) ) * scl(i+1,j,d) &
             +  ( - 1.0_RP * coef(i,j,d,1) &
                  - 1.0_RP * coef(i,j,d,2) &
                  + 2.0_RP * coef(i,j,d,3) ) * scl(i,j+1,d) &
           ) / 3.0_RP
enddo
enddo
enddo
```

Listing 10 Example of the diffusion operator ported to GridTools

```
template <typename evaluation>
  GT_FUNCTION
  static void Do(evaluation const & eval, x_interval) {
    eval(vt{}) =  (( + 2.0 * eval(coef{})
                     - 1.0 * eval(coef{a+1})
                     - 1.0 * eval(coef{a+2}) ) * eval(scl{})
                +  ( - 1.0 * eval(coef{})
                     + 2.0 * eval(coef{a+1})
                     - 1.0 * eval(coef{a+2}) ) * eval(scl{i+1})
                +  ( - 1.0 * eval(coef{})
                     - 1.0 * eval(coef{a+1})
                     + 2.0 * eval(coef{a+2}) ) * eval(scl{j+1})
                ) / 3.0;
}
```

5 Massive I/O and Compression

5.1 *The Scientific Compression Library (SCIL)*

The developed compression library SCIL [36] provides a framework to compress structured and unstructured data using the best available (lossless or lossy) compres-

www.dbooks.org

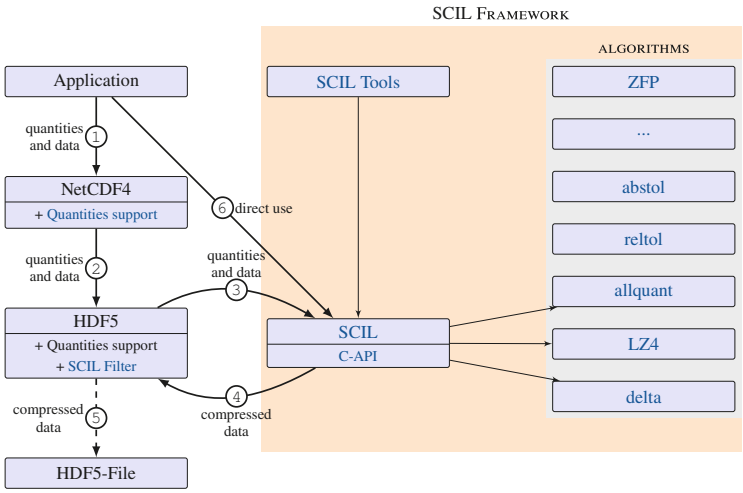


Fig. 8 SCIL compression path and components (extended)[36]

sion algorithms according to the definition of tolerable loss of accuracy and required performance. SCIL acts as a meta-compressor providing various backends such as algorithms like LZ4, ZFP, SZ but also integrates some alternative algorithms.

The data path of SCIL is illustrated in Fig. 8. An application can either use NetCDF4 [45],[2] HDF5 [19] or directly the C-interface of SCIL. Based on the defined quantities, the values and the characteristics of the data to compress, the appropriate compression algorithm is chosen. SCIL also comes with a library to generate various synthetic test patterns for compression studies, i.e., well-defined multi-dimensional data patterns of any size. Further tools are provided to plot, to add noise or to compress CSV and NetCDF3 files.

5.2 *Supported Quantities*

There are three types of quantities supported:

Accuracy Quantities define the tolerable error on lossy compression. When compressing the value v to $\hat{v}$ it bounds the residual error ($r = v - \hat{v}$):

- **absolute tolerance**: $v - \text{abstol} \leq \hat{v} \leq v + \text{abstol}$
- **relative tolerance**: $v/(1 + reltol) \leq \hat{v} \leq v \cdot (1 + reltol)$

[2]HDF5 and NetCDF4 are APIs and self-describing data formats for storing multi-dimensional data with user-relevant metadata.

- **relative error finest tolerance**: used together with relative tolerance; absolute tolerable error for small v's. If relfinest $> |v \cdot (1 \pm reltol)|$, then $v -$ relfinest $\leq \hat{v} \leq v +$ relfinest
- **significant digits**: number of significant decimal digits
- **significant bits**: number of significant digits in bits

SCIL must ensure that all the set accuracy quantities are honored regardless of the algorithm chosen, meaning that one can set, e.g., absolute and relative tolerance and the strictest of the criteria is satisfied.

Performance Quantities These quantities define the expected performance behavior for both compression and decompression (on the same system). The value can be defined according to: (1) absolute throughput in MiB or GiB; or (2) relative to network or storage speed. It is considered to be the expected performance for SCIL but it may not be as strictly handled as the qualities—there may be some cases in which performance is lower. Thus, SCIL must estimate the compression rates for the data.

Supplementary Quantities An orthogonal quantity that can be set is the so called *fill value*, a value that scientists use to mark special data points. This value must be preserved accurately and usually is a specific high or low value that may disturb a smooth compression algorithm.

5.3 Compression Chain

Internally, SCIL creates a compression chain which can involve several compression algorithms as illustrated in Fig. 9. Based on the basic datatype that is supplied, the initial stage of the chain is entered. Algorithms may be preconditioners to optimize data layout for subsequent compression algorithms, converters from one data format to another, or, on the final stage, a lossless compressor. Floating-point data can be first mapped to integer data and then to a byte stream. Intermediate steps can be skipped.

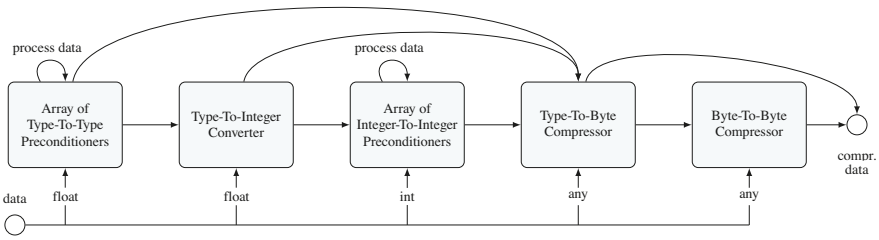


Fig. 9 SCIL compression chain. The data path depends on the input data type [36]

www.dbooks.org

5.4 Algorithms

SCIL comes with additional algorithms that are derived to support one or multiple accuracy quantities set by the user. For example, the algorithms Abstol (for absolute tolerance), Sigbits (for significant bits/digits), and Allquant. These algorithms aim to pack the number of required bits as tightly as possible into the data buffer but operate on each value independently. While Abstol and Sigbits just consider one quantity, Allquant considers all quantities together and chooses the required operation for a data point depending on the highest precision needed. We also consider these algorithms to be useful baselines when comparing any other algorithm. ZFP and SZ, for example, work on one quantity, too.

During the project, we explored the implementation for the automatic algorithm selection but only integrated a trivial scheme for the following reasons: if only a single quantity is set, we found out that the optimal parameter depends on many factors (features); the resulting optimal choice is embedded in a multi-dimensional space—this made it infeasible to identify the optimal algorithm. Once more than a single quantity is set, only one of the newly integrated algorithms can perform the compression, which eliminates any choice. As the decoupling of SCIL enables to integrate algorithms in the future, we hope that more algorithms will be developed that can then benefit from implementing the automatic selection.

6 Evaluation of the Compression Library SCIL

We evaluated the performance and compression ratio of SCIL against several scientific data sets and the synthetic test patterns generated by SCIL itself [36].

6.1 Single Core Performance

In the following, an excerpt of the experiments conducted with SCIL on a single core is shown. These results help to understand the performance behavior of compression algorithms and their general characteristics.

Four data sets were used each with precision floating-point data (32 bit): (1) the data created with the SCIL pattern library (10 data sets each with different random seed numbers). The synthetic data has the dimensionality of ($300 \times 300 \times 100 =$ 36 MB); (2) the output of the ECHAM atmospheric model [46] which stored 123 different scientific variables for a single timestep as NetCDF; (3) the output of the hurricane Isabel model which stored 633 variables for a single timestep as binary;[3]

Table 5 Harmonic mean compression performance for different scientific data sets

	Algorithm	Ratio	Compr. MiB/s	Decomp. MiB/s
(*a*) 1% *absolute tolerance*				
NICAM	abstol	0.206	499	683
	abstol,lz4	0.015	458	643
	sz	0.008	122	313
	zfp-abstol	0.129	302	503
ECHAM	abstol	0.190	260	456
	abstol,lz4	0.062	196	400
	sz	0.078	81	169
	zfp-abstol	0.239	185	301
Isabel	abstol	0.190	352	403
	abstol,lz4	0.029	279	356
	sz	0.016	70	187
	zfp-abstol	0.039	239	428
Random	abstol	0.190	365	382
	abstol,lz4	0.194	356	382
	sz	0.242	54	125
	zfp-abstol	0.355	145	241
(*b*) 9 *bits precision*				
NICAM	sigbits	0.439	257	414
	sigbits,lz4	0.216	182	341
	zfp-precision	0.302	126	182
ECHAM	sigbits	0.448	462	615
	sigbits,lz4	0.228	227	479
	zfp-precision	0.299	155	252
Isabel	sigbits	0.467	301	506
	sigbits,lz4	0.329	197	366
	zfp-precision	0.202	133	281
Random	sigbits	0.346	358	511
	sigbits,lz4	0.348	346	459
	zfp-precision	0.252	151	251

(4) the output of the NICAM Icosahedral Global Atmospheric model which stored 83 variables as NetCDF.

The characteristics of the scientific data varies and so does data locality within the data sets. For example, in the Isabel data many variables are between 0 and 0.02, many between −80 and +80 and some are between −5000 and 3000.

We set only one quantity to allow using ZFP and SZ for comparison. Table 5 shows the harmonic mean compression ratio[4] for setting an absolute error of 1% of

[3] http://vis.computer.org/vis2004contest/data.html.

[4] The ratio is the resulting file size divided by the original file size.

www.dbooks.org

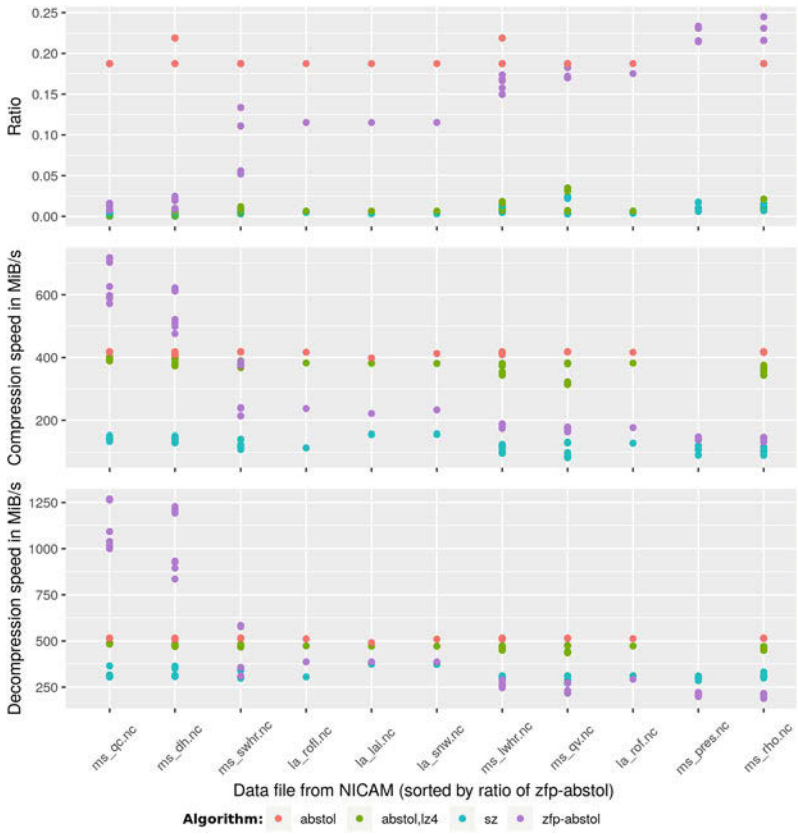


Fig. 10 Compressing various climate variables with absolute tolerance 1%

the maximum value or setting precision to 9 bit accuracy. The harmonic mean corresponds to the total reduction and performance when compressing/decompressing all the data.

The results for compressing 11 variables of the whole NICAM model via the NetCDF API are shown in Fig. 10. The x-axis represents the different data files, as each file consists of several chunks, a point in the y-axis represents one chunk. It can be observed that generally the SZ algorithm yields the best compression ratio but Abstol+LZ4 yields the second best ratio providing much better and predictable compression and decompression speeds.

Note that for some individual variables, one algorithm may supersede another in terms of ratio. As expected there are cases in which one algorithm is outperforming the other algorithms in terms of compression ratio which justifies the need for a metacompressor like SCIL that can make smart choices on behalf of the users. Some algorithms perform generally better than others in terms of performance. Since in

our use cases, users define the tolerable error, we did not investigate metrics that compare the precision for a fixed compression ratio (e.g., the signal to noise ratio).

While a performance of 200 MB/s may look insufficient for a single core, with 24 cores per node a good speed per node can be achieved that is still beneficial for medium large runs on shared storage. For instance, consider a storage system that can achieve 500 GB/s. Considering that one node with typical Infiniband configuration can transfer at least 5 GB/s, 100 client nodes saturate the storage. By compressing 5:1 (or ratio of 0.2), virtually, the storage could achieve a peak performance of 2500 GB/s, and, thus, can serve up to 500 client nodes with (theoretical) maximum performance.

Trading of storage capacity vs. space is an elementary issue to optimize bigger workflows. By separating the concerns between the necessary data quality as defined by scientists and compression library, site-specific policies could be employed that depend also on the available hardware.

6.2 Compression in HDF5 and NetCDF

We tested the compression library SCIL using the icosahedral grid code. The code can use NetCDF to output the generated data periodically. In this experiment, a high-resolution grid with 268 million grid cells (single precision floating point) in the horizontal times 64 vertical levels was used and executed on the supercomputer Mistral. The code was run on 128 nodes, with one MPI process per node. It wrote one field to the output file in one timestep. The field values range between -10 to $+55$ which is important for understanding the impact of the compression.

The experiments varied the basic compression algorithms and parameters provided by SCIL. Compression is done with the algorithms

- memcopy: does not do any real compression, but allows to measure the overhead of the usage of enabling HDF5 compression and SCIL.
- lz4: the well-known compression algorithm. It is unable to compress floating-point data but slows down the execution.
- abstol,lz4: processes data elements based on the absolute value of each point, we control the tolerance by the parameter $absolute_tolerance$, after quantization an LZ4 compression step is added.
- sigbits,lz4: processes data elements based on a percentage of the value being processed, we control the tolerance by the parameter $relative_tolerance_percent$, after quantization an LZ4 compression step is added.

The results of this selection of compression methods is shown in Table 6, it shows the write time in seconds and resulting data size in GB, a virtual throughput relative to the uncompressed file size, and the speedup. Without compression the performance is quite poor: achieving only 432 MB/s on Mistral on 128 nodes, while an optimal benchmark can achieve 100 GB/s. The HDF5 compression is not yet optimally parallelized and requires certain collective operations to update

www.dbooks.org

Table 6 Compression results of 128 processes on Mistral

Compression method	Parameter	Write time in s	Data size in GB	Throughput* in MB/s	Speedup
No-compression		165.3	71.4	432	1.0
memcopy		570.1	71.4	125	0.3
lz4		795.3	71.9	90	0.2
abstol,lz4	absolute_tolerance=1	12.8	2.7	5578	12.9
	absolute_tolerance=.1	72.6	2.8	983	2.3
sigbits,lz4	relative_tolerance_percent=1	12.9	2.9	5535	12.8
	relative_tolerance_percent=.1	18.3	3.2	3902	9.0

the metadata. Internally, HDF5 requires additional data buffers. This leads to extra overhead in the compression slowing down the I/O (see the memcopy and and LZ4 results which serve as baselines). By activating lossy compression and accepting an accuracy of 1% or 0.1%, the performance can be improved in this example up to 13x.

Remember that these results serve as feasibility study. One of our goals was to provide a NetCDF backwards compatible compression method not to optimize the compression data path inside HDF5. The SCIL library can be used directly by existing models avoiding the overhead and leading to the results as shown above.

7 Standardized Icosahedral Benchmarks

Our research on DSL and I/O is more practical. We started with real-world applications, namely three global atmospheric models developed by the participating countries. The global atmospheric model with the icosahedral grid system is one of the new generation global climate/weather models. The grid-point calculations, which are less computational intense than the spherical harmonics transformation, are used in the model. On the other hand, patterns of the data access in differential operators are more complicated than the traditional limited-area atmospheric model with the Cartesian grid system.

There are different implementations of the dynamical core on the icosahedral grid: direct vs. indirect memory access, staggered vs. co-located data distribution, and so on. The objective of standardization of benchmarks is to provide a variety of computational patterns of the icosahedral atmospheric models. The kernels are useful for evaluating the performance of new machines and new programming models. Not only for our studies but also for the existing/future DSL studies, this benchmark set provides various samples of the source code. The icosahedral grid system is an unstructured grid coordinate, and there are a lot of challenging issues about data decomposition, data layout, loop structures, cache blocking, threading, offloading the accelerators, and so on. By applying DSLs or frameworks to the kernels, the developers can try the detailed, practical evaluation of their software.

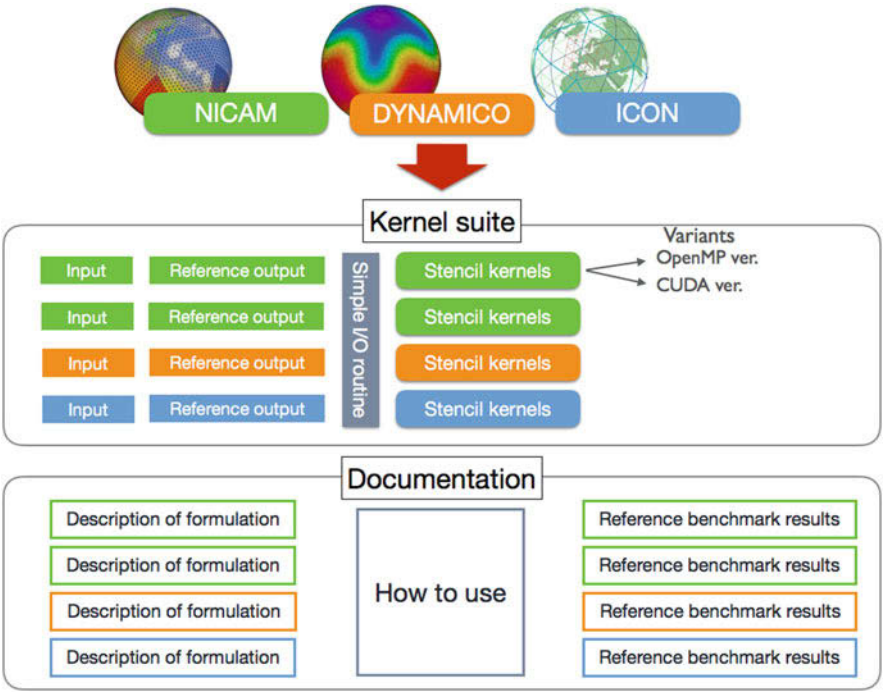


Fig. 11 Overview of IcoAtmosBenchmark v1

The benchmarks were used in the final evaluation of SCIL and they steered the DSL development by providing the relevant patterns.

7.1 IcoAtmosBenchmark v1: Kernels from Icosahedral Atmospheric Models

IcoAtmosBenchmark v1 is the package of kernels extracted from three Icosahedral Global Atmospheric models, NICAM, ICON, DYNAMICO. As shown in Fig. 11, we prepared input data and reference output data for easy validation of results. We also arranged documentation about the kernels. The package is available online.[5]

[5] https://aimes-project.github.io/IcoAtmosBenchmark_v1/.

www.dbooks.org

7.1.1 Documentation

An excerpt to the NICAM kernel serves as an example. The icosahedral grid on the sphere of NICAM is the unstructured grid system. In NICAM code, the complex topology of the grid system is separated into the structured and unstructured part. The grids are decomposed into tiles, and one or more tiles are allocated to each MPI process. The horizontal grids in the tile are kept in a 2-dimensional structure. On the other hand, a topology of the network of the tiles is complex. We selected and extracted 6+1 computational kernels from the dynamical core of NICAM, as samples of the stencil calculation on the structured grid system. We also extracted a communication kernel, as a sample of halo exchange in the complex node topology. The features of each kernel are documented on the GitHub page.

All kernels are single subroutines and almost the same as the source codes in the original model, except for the ICON kernels. They are put into the wrapper for the kernel program. Values of input variables in the argument list of the kernel subroutine are stored as a data file, just before the call in the execution of the original model. They are read and given to the subroutine in the kernel program. Similarly, the values of output variables in the argument list are stored, just after the call in execution. They are read and compared to the actual output values of kernel subroutine. The differences are written to the standard output for validation. For easy handling of the input/reference data by both the Fortran program and C program, we prepared an I/O routine written in C.

We provided a user manual, which contains the brief introduction of each model, the description of each kernel, usage of kernel programs, and sample results. This information is helpful for users of this kernel suite in the future.

8 Summary and Conclusion

The numerical simulation of climate and weather is demanding for computational and I/O resources. Within the AIMES project, we addressed those challenges and researched approaches that foster the separation of concerns. This idea unites our approaches for the DSL and the compression library. While a higher level of abstraction can improve the productivity for scientists, most importantly the decoupling of requirements from the implementation allows scientific programmers to develop and improve architecture-specific optimizations.

Promising candidates for DSLs have been explored and with GGDML an alternative has been developed that covers the most relevant formulations of the three icosahedral models: DYNAMICO, ICON, and NICAM. The DSL and toolchain we developed integrates into existing code bases and suits for incremental reformulation of the code. We estimated the benefit for code reduction and demonstrated several optimizations for CPU, GPU, and vector architectures. Our DSL allows to reduce code to around 30% of the LOC in comparison to code written with GPL code.

With the semantics of GGDML, we could achieve near optimal use of memory hierarchies and memory throughput which is critical for the family of computations in hand. Our experiments show running codes with around 80% of achievable memory throughput on different archiectures. Furthermore, we could scale models to multiple nodes, which is essential for exascale computing, using the same code that is used for a single node. The separation of concerns in our approach allowed us to keep models code separate of underlying hardware changes. The single GGDML source code is used to generate code for the different architectures and on single vs. multiple nodes.

To address the I/O challenge, we developed the library SCIL, a metacompressor supporting various lossy and lossless algorithms. It allows users to specify various quantities for the tolerable error and expected performance, and allows the library to chose a suitable algorithm. SCIL is a stand-alone library but also integrates into NetCDF and HDF5 allowing users to explore the benefits of using alternative compression algorithms with their existing middleware. We evaluated the performance and compression ratio for various scientific data sets and on moderate scale. The results show that the choice of the best algorithm depends on the data and performance expectation which, in turn, motivates the need for the decoupling of quantities from the selection of the algorithm. A blocker for applying parallel large-scale compression in existing NetCDF workflows is the performance limitation of the current HDF5 stack.

Finally, benchmarks and mini-applications were created that represent the key features of the icosahedral applications.

Beside the achieved research in the AIMES project, the work done opens the door for further research in the software engineering of climate and weather prediction models. The performance portability, where we used the same code to run on different architectures and machines, including single and multiple nodes, shows that techniques are viable to continue the research in this direction. The code reduction offered by DSLs promises to save millions in development cost that can be used to contribute to the DSL development. During the runtime of the project, it became apparent that the concurrently developed solutions GridTools and PSYclone that also provide such features are preferred by most scientists as they are developed by a bigger community and supported by national centers. We still believe that the developed light-weight DSL solution provides more flexibility particularly for smaller-sized models and can be maintained as part of the development of the models itself. It also can be used in other contexts providing domain-specific templates to arbitrary codes.

In the future, we aim to extend the DSL semantics to also address I/O relevant specifications. This would allow to unify the effort towards optimal storage and computation.

www.dbooks.org

References

1. Adams, S.V., Ford, R.W., Hambley, M., Hobson, J., Kavčič, I., Maynard, C., Melvin, T., Müller, E.H., Mullerworth, S., Porter, A., et al.: LFRic: Meeting the challenges of scalability and performance portability in Weather and Climate models. J. Parallel Distrib. Comput. **132**, 383–396 (2019)
2. Alforov, Y., Novikova, A., Kuhn, M., Kunkel, J., Ludwig, T.: Towards green scientific data compression through high-level I/O interfaces. In: 30th International Symposium on Computer Architecture and High Performance Computing, pp. 209–216. Springer, Berlin (2019). https://doi.org/10.1109/CAHPC.2018.8645921
3. Baker, A.H., Xu, H., Dennis, J.M., Levy, M.N., Nychka, D., Mickelson, S.A., Edwards, J., Vertenstein, M., Wegener, A.: A methodology for evaluating the impact of data compression on climate simulation data. In: Proceedings of the 23rd international symposium on High-performance parallel and distributed computing, pp. 203–214. ACM, New York (2014)
4. Baker, A.H., Hammerling, D.M., Mickelson, S.A., Xu, H., Stolpe, M.B., Naveau, P., Sanderson, B., Ebert-Uphoff, I., Samarasinghe, S., De Simone, F., Gencarelli, C.N., Dennis, J.M., Kay, J.E., Lindstrom, P.: Evaluating lossy data compression on climate simulation data within a large ensemble. Geosci. Model Dev. **9**, 4381–4403 (2016). https://doi.org/10.5194/gmd-9-4381-2016
5. Baron, T.J., Khlopkov, K., Pretorius, T., Balzani, D., Brands, D., Schröder, J.: Modeling of microstructure evolution with dynamic recrystallization in finite element simulations of martensitic steel. Steel Res. Int. **87**(1), 37–45 (2016)
6. Barry, B., et al.: Software Engineering Economics, vol. 197. Prentice-Hall, New York (1981)
7. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., et al.: Advances concerning multiscale methods and uncertainty quantification in EXA-DUNE. In: Software for Exascale Computing-SPPEXA 2013-2015, pp. 25–43. Springer, Berlin (2016)
8. Bauer, P., Thorpe, A., Brunet, G.: The quiet revolution of numerical weather prediction. Nature **525**(7567), 47 (2015)
9. Bauer, S., Drzisga, D., Mohr, M., Rüde, U., Waluga, C., Wohlmuth, B.: A stencil scaling approach for accelerating matrix-free finite element implementations. SIAM J. Sci. Comput. **40**(6), C748–C778 (2018)
10. Bauer, S., Huber, M., Ghelichkhan, S., Mohr, M., Rüde, U., Wohlmuth, B.: Large-scale simulation of mantle convection based on a new matrix-free approach. J. Comput. Sci. **31**, 60–76 (2019)
11. Bicer, T., Agrawal, G.: A compression framework for multidimensional scientific datasets. In: 2013 IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW), pp. 2250–2253 (2013). https://doi.org/10.1109/IPDPSW.2013.186
12. CSCS Claw. http://www.xcalablemp.org/download/workshop/4th/Valentin.pdf
13. CSCS GridTools. https://github.com/GridTools/gridtools
14. DeVito, Z., Joubert, N., Palacios, F., Oakley, S., Medina, M., Barrientos, M., Elsen, E., Ham, F., Aiken, A., Duraisamy, K., et al.: Liszt: a domain specific language for building portable mesh-based pde solvers. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 9. ACM, New York (2011)
15. Di, S., Cappello, F.: Fast Error-bounded Lossy HPC data compression with SZ. In: 2016 IEEE International Parallel and Distributed Processing Symposium, pp. 730–739. IEEE, Piscataway (2016). https://doi.org/10.1109/IPDPS.2016.11
16. Dolbeau, R., Bihan, S., Bodin, F.: Hmpp: a hybrid multi-core parallel programming environment. In: Workshop on General Purpose Processing on Graphics Processing Units (GPGPU 2007), vol. 28 (2007)
17. Dubos, T., Dubey, S., Tort, M., Mittal, R., Meurdesoif, Y., Hourdin, F.: Dynamico, an icosahedral hydrostatic dynamical core designed for consistency and versatility. Geosci. Model Dev. Discuss. **8**(2) (2015)

18. Faghih-Naini, S., Kuckuk, S., Aizinger, V., Zint, D., Grosso, R., Köstler, H.: Towards whole program generation of quadrature-free discontinuous Galerkin methods for the shallow water equations. arXiv preprint:1904.08684 (2019)
19. Folk, M., Cheng, A., Yates, K.: HDF5: a file format and I/O library for high performance computing applications. In: Proceedings of Supercomputing, vol. 99, pp. 5–33 (1999)
20. Gerbes, A., Kunkel, J., Jumah, N.: Intelligent Selection of Compiler Options to Optimize Compile Time and Performance (2017). http://llvm.org/devmtg/2017-03//2017/02/20/accepted-sessions.html#42
21. Gerbes, A., Jumah, N., Kunkel, J.: Automatic Profiling for Climate Modeling (2018). http://llvm.org/devmtg/2017-03//2017/02/20/accepted-sessions.html#42
22. Gomez, L.A.B., Cappello, F.: Improving floating point compression through binary masks. In: 2013 IEEE International Conference on Big Data (2013). https://doi.org/10.1109/BigData.2013.6691591
23. Gysi, T., Fuhrer, O., Osuna, C., Cumming, B., Schulthess, T.: Stella: A domain-specific embedded language for stencil codes on structured grids. In: EGU General Assembly Conference Abstracts, vol. 16 (2014)
24. Heene, M., Hinojosa, A.P., Obersteiner, M., Bungartz, H.J., Pflüger, D.: EXAHD: an Exa-Scalable two-level sparse grid approach for higher-dimensional problems in plasma physics and beyond. In: High Performance Computing in Science and Engineering'17, pp. 513–529. Springer, Berlin (2018)
25. Hübbe, N., Kunkel, J.: Reducing the HPC-Datastorage footprint with MAFISC—Multidimensional Adaptive Filtering Improved Scientific data Compression. In: Computer Science—Research and Development, pp. 231–239 (2013). https://doi.org/10.1007/s00450-012-0222-4
26. Hübbe, N., Wegener, A., Kunkel, J., Ling, Y., Ludwig, T.: Evaluating lossy compression on climate data. In: Kunkel, J.M., Ludwig, T., Meuer, H.W. (eds.) Supercomputing, no. 7905 in Lecture Notes in Computer Science, pp. 343–356. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-38750-0_26
27. Iverson, J., Kamath, C., Karypis, G.: Fast and effective lossy compression algorithms for scientific datasets. In: Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7484, pp. 843–856. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-32820-6_83
28. Jum'ah, N., Kunkel, J.: Automatic vectorization of stencil codes with the GGDML language extensions. In: Workshop on Programming Models for SIMD/Vector Processing (WPMVP'19), February 16, 2019, Washington, DC, USA, WPMVP, pp. 1–7. PPoPP 2019. ACM, New York (2019).https://doi.org/10.1145/3303117.3306160. https://ppopp19.sigplan.org/home/WPMVP-2019
29. Jumah, N., Kunkel, J.: Optimizing memory bandwidth efficiency with user-preferred kernel merge. In: Lecture Notes in Computer Science. Springer, Berlin (2019)
30. Jum'ah, N., Kunkel, J.: Performance portability of earth system models with user-controlled GGDML code translation. In: Yokota, R., Weiland, M., Shalf, J., Alam, S. (eds.) High Performance Computing: ISC High Performance 2018 International Workshops, Frankfurt/Main, Germany, June 28, 2018, Revised Selected Papers, no. 11203. Lecture Notes in Computer Science, pp. 693–710. ISC Team, Springer, Berlin (2019). https://doi.org/10.1007/978-3-030-02465-9_50
31. Jumah, N., Kunkel, J.: Scalable Parallelization of Stencils using MODA. In: High Performance Computing: ISC High Performance 2019 International Workshops, Frankfurt/Main, Germany, June 20, 2019, Revised Selected Papers. Lecture Notes in Computer Science. Springer, Berlin (2019)
32. Jumah, N., Kunkel, J., Zängl, G., Yashiro, H., Dubos, T., Meurdesoif, Y.: GGDML: Icosahedral models language extensions. J. Comput. Sci. Technol. Updates **4**(1), 1–10 (2017). https://doi.org/10.15379/2410-2938.2017.04.01.01.http://www.cosmosscholars.com/images/JCSTU_V4N1/JCSTU-V4N1A1-Jumah.pdf

www.dbooks.org

33. Kronawitter, S., Kuckuk, S., Köstler, H., Lengauer, C.: Automatic data layout transformations in the ExaStencils code generator. Mod. Phys. Lett. A **28**(03), 1850009 (2018)
34. Kunkel, J.: Analyzing data properties using statistical sampling techniques—illustrated on scientific file formats and compression features. In: Taufer, M., Mohr, B., Kunkel, J. (eds.) High Performance Computing: ISC High Performance 2016 International Workshops, ExaComm, E-MuCoCoS, HPC-IODC, IXPUG, IWOPH, P3MA, VHPC, WOPSSS, no. 9945. Lecture Notes in Computer Science, pp. 130–141. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-46079-6_10
35. Kunkel, J.: SFS: A Tool for Large Scale Analysis of Compression Characteristics. Technical Report 4, Deutsches Klimarechenzentrum GmbH, Bundesstraße 45a, D-20146 Hamburg (2017)
36. Kunkel, J., Novikova, A., Betke, E.: Towards Decoupling the Selection of Compression Algorithms from Quality Constraints—an Investigation of Lossy Compression Efficiency. Supercomputing Front. Innov. **4**(4), 17–33 (2017). https://doi.org/10.14529/jsfi170402. http://superfri.org/superfri/article/view/149
37. Kunkel, J., Novikova, A., Betke, E., Schaare, A.: Toward decoupling the selection of compression algorithms from quality constraints. In: Kunkel, J., Yokota, R., Taufer, M., Shalf, J. (eds.) High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P3MA, VHPC, Visualization at Scale, WOPSSS. No. 10524 in Lecture Notes in Computer Science, pp. 1–12. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-67630-2_1
38. Lakshminarasimhan, S., Shah, N., Ethier, S., Klasky, S., Latham, R., Ross, R., Samatova, N.: Compressing the incompressible with ISABELA: In-situ reduction of spatio-temporal data. In: European Conference on Parallel and Distributed Computing (Euro-Par), Bordeaux, France (2011). https://doi.org/10.1007/978-3-642-23400-2_34
39. Laney, D., Langer, S., Weber, C., Lindstrom, P., Wegener, A.: Assessing the effects of data compression in simulations using physically motivated metrics. Super Comput. (2013). https://doi.org/10.3233/SPR-140386
40. Lindstrom, P.: Fixed-rate compressed floating-point arrays. In: IEEE Transactions on Visualization and Computer Graphics 2012 (2014). https://doi.org/10.1109/BigData.2013.6691591
41. Lindstrom, P., Isenburg, M.: Fast and efficient compression of floating-point data. IEEE Trans. Vis. Comput. Graph. **12**(5), 1245–1250 (2006). https://doi.org/10.1109/TVCG.2006.143
42. Maruyama, N., Sato, K., Nomura, T., Matsuoka, S.: Physis: an implicitly parallel programming model for stencil computations on large-scale GPU-accelerated supercomputers. In: 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–12. IEEE, Piscataway (2011)
43. Müller, M., Aoki, T.: Hybrid fortran: high productivity GPU porting framework applied to Japanese weather prediction model. arXiv preprint: 1710.08616 (2017)
44. Rathgeber, F., Ham, D.A., Mitchell, L., Lange, M., Luporini, F., McRae, A.T., Bercea, G.T., Markall, G.R., Kelly, P.H.: Firedrake: automating the finite element method by composing abstractions. ACM Trans. Math. Softw. (TOMS) **43**(3), 24 (2017)
45. Rew, R., Davis, G.: NetCDF: an interface for scientific data access. IEEE Comput. Graph. Appl. **10**(4), 76–82 (1990)
46. Roeckner, E., Bäuml, G., Bonaventura, L., Brokopf, R., Esch, M., Giorgetta, M., Hagemann, S., Kirchner, I., Kornblueh, L., Manzini, E., et al.: The Atmospheric General Circulation Model ECHAM 5. Part I: Model Description. Report/Max-Planck-Institut für Meteorologie, p. 349, (2003). http://hdl.handle.net/11858/00-001M-0000-0012-0144-5
47. Satoh, M., Tomita, H., Yashiro, H., Miura, H., Kodama, C., Seiki, T., Noda, A.T., Yamada, Y., Goto, D., Sawada, M., et al.: The non-hydrostatic icosahedral atmospheric model: description and development. Prog. Earth. Planet. Sci. **1**(1), 18 (2014)
48. Thaler, F., Moosbrugger, S., Osuna, C., Bianco, M., Vogt, H., Afanasyev, A., Mosimann, L., Fuhrer, O., Schulthess, T.C., Hoefler, T.: Porting the cosmo weather model to manycore CPUS. In: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC '19, pp. 13:1–13:11 (2019)

49. Tietz, J.: Vector Folding for Icosahedral Earth System Models (2018). https://wr.informatik.uni-hamburg.de/_media/research:theses:jonas_tietz_vector_folding_for_icosahedral_earth_system_models.pdf
50. Torres, R., Linardakis, L., Kunkel, T.J., Ludwig, T.: Icon dsl: a domain-specific language for climate modeling. In: International Conference for High Performance Computing, Networking, Storage and Analysis, Denver, Colo (2013). http://sc13.supercomputing.org/sites/default/files/WorkshopsArchive/track139.html
51. Unat, D., Cai, X., Baden, S.B.: Mint: realizing cuda performance in 3D stencil methods with annotated c. In: Proceedings of the International Conference on Supercomputing, pp. 214–224. ACM, New York (2011)
52. van Engelen, R.A.: Atmol: a domain-specific language for atmospheric modeling. CIT. J. Comput. Inf. Technol. **9**(4), 289–303 (2001)
53. Velten, K.: Mathematical Modeling and Simulation: Introduction for Scientists and Engineers. Wiley, New York (2009)
54. Zängl, G., Reinert, D., Rípodas, P., Baldauf, M.: The icon (icosahedral non-hydrostatic) modelling framework of dwd and mpi-m: description of the non-hydrostatic dynamical core. Q. J. R. Meteorol. Soc. **141**(687), 563–579 (2015)
55. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Trans. Inf. Theory **23**(3), 337–343 (1977). https://doi.org/10.1109/TIT.1977.1055714

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

DASH: Distributed Data Structures and Parallel Algorithms in a Global Address Space



Karl Fürlinger, José Gracia, Andreas Knüpfer, Tobias Fuchs, Denis Hünich, Pascal Jungblut, Roger Kowalewski, and Joseph Schuchart

Abstract DASH is a new programming approach offering distributed data structures and parallel algorithms in the form of a C++ template library. This article describes recent developments in the context of DASH concerning the ability to execute tasks with remote dependencies, the exploitation of dynamic hardware locality, smart data structures, and advanced algorithms. We also present a performance and productivity study where we compare DASH with a set of established parallel programming models.

1 Introduction

DASH is a parallel programming approach that realizes the PGAS (partitioned global address space) model and is implemented as a C++ template library. DASH tries to reconcile the productivity advantages of shared memory parallel programming with the physical realities of distributed memory hardware. To achieve this goal, DASH provides the abstraction of globally accessible memory that spans multiple interconnected nodes. For performance reasons, this global memory is partitioned and data locality is not hidden but explicitly exploitable by the application developer.

DASH is realized as a C++ template library, obviating the need for a custom language and compiler. By relying on modern C++ abstraction and implementation techniques, a productive programming environment can be built solely based on

K. Fürlinger (✉) · T. Fuchs · P. Jungblut · R. Kowalewski
Ludwig-Maximilians-Universität München, Munich, Germany
e-mail: Karl.Fuerlinger@ifi.lmu.de

J. Gracia · J. Schuchart
University of Stuttgart, Stuttgart, Germany

A. Knüpfer · D. Hünich
TU Dresden, Dresden, Germany

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_6

standard components. For many application developers in HPC and in general, a big part of the appeal of the C++ programming language stems from the availability of high performance generic data structures and algorithms in the C++ standard template library (STL).

DASH can be seen as a generalization of concepts found in the STL to the distributed memory case and efforts have been made to keep DASH compatible with components of the STL. In many cases it is thus possible to mix and match algorithms and data structures freely between DASH and the STL.

DASH is developed in the context of the SPPEXA priority programme for Exascale computing since 2013. In this paper we give an overview of DASH and report on activities within the project focusing on the second half of the funding period. We first give an overview of the DASH Runtime System (DART) in Sect. 2, focusing on features related to task execution with global dependencies and dynamic hardware topology discovery. In Sect. 3 we describe two components of the DASH C++ template library, a smart data structure that offers support for productive development of stencil codes and an efficient implementation of parallel sorting. In Sect. 4 we provide an evaluation of DASH testing the feasibility of our approach. We provide an outlook on future developments in Sect. 5.

2 The DASH Runtime System

The DASH Runtime System (DART) is implemented in C and provides an abstraction layer on top of distributed computing hardware and one-sided communication substrates. The main functionality provided by DART is memory allocation and addressing as well as communication in a global address space. In DASH parlance the individual participants in an application are called *units* mapped to MPI processes in the MPI-3 remote memory access based implementation of DART.

Early versions of DASH/DART focused on data distribution and access and offered no explicit compute model. This has changed with the support for tasks in DASH and DART. We start with a discussion of these new features, followed by a description of efforts to tackle increasing hardware complexity in Sect. 2.2.

2.1 Tasks with Global Dependencies

The benefit of decoupled transfer and synchronization in the PGAS programming model promises to provide improved scalability and better exploit hardware capabilities. However, proper synchronization of local and global memory accesses is essential for the development of correct applications. So far, the synchronization constructs in DASH were limited to collective synchronization using barriers and reduction operations as well as an implementation of the MCS lock. Using atomic RMA operations, users could also create custom synchronization schemes using

www.dbooks.org

point-to-point signaling, i.e., by setting a flag at the target after completion of a transfer. While this approach might work for simple examples, it hardly scales to more complex examples where reads and writes from multiple processes need to be synchronized.

The need for a more fine-grained way of synchronization that allows to create more complex synchronization patterns was thus imminent. The data-centric programming model of DASH with the distributed data structure at its core lead motivated us to create a synchronization that centers around these global data structures, i.e., which is data-centric itself. At the same time, the essential property of PGAS needed to be preserved: the synchronization had to remain decoupled from data transfers, thus not forcing users to rely solely on the new synchronization mechanism for data transfers.

At the same time, the rise of task-based programming models inspired us to investigate the use of tasks as a synchronization vehicle, i.e., by encapsulating local and global memory accesses into tasks that are synchronized using a data-centric approach. Examples of widely known data-centric task-based synchronization models are OpenMPI with its task data dependencies, OmpSs, and PaRSEC. While PaRSEC uses data dependencies to express both synchronization and actual data flow between tasks, OpenMP and OmpSs use data dependencies solely for synchronization without affecting data movement. In contrast to PaRSEC, however, OpenMP and OmpSs only support shared memory parallelization.

A different approach has been taken by HPX, which facilitates synchronization through the use of future/promise pairs, which form a channel between two or more tasks and are a concept that has been well established in the C++ community. However, this synchronization concept with it's inherent communication channel hardly fits into the concept of a PGAS abstraction built around data structures in the global memory space. Moreover, DASH provides a locality-aware programming, in which processes know their location in the global address and can diverge their control accordingly, whereas HPX is a locality-agnostic programming model.

We thus decided to focus our research efforts on distributed data dependencies, extending the shared memory capabilities of task data dependencies into the global memory space while keeping synchronization and data transfer decoupled.

2.1.1 Distributed Data Dependencies

Tasks in DASH are created using the `async` function call and passing it an action that will be executed by a worker thread at a later point in time. Additionally, the `async` function accepts an arbitrary number of dependency definitions of the form `in(memory_location)` and `out(memory_location)` to define input and output dependencies, respectively. In the DASH tasking model, each unit discovers it's local task graph by creating tasks operating mainly in the local portion of the global memory space, i.e., tasks are never transferred to other units. This locality-awareness limits the number of tasks to be discovered to only the tasks that will eventually be executed in that unit: as depicted in Fig. 1.

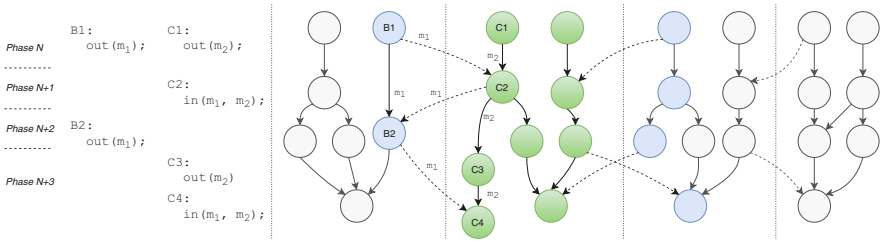


Fig. 1 Distributed task graph discovery: the unit in the middle only discovers its local tasks (green) and should only be concerned with tasks on other units that have dependencies to its local tasks (blue). All other tasks (gray) should not be considered

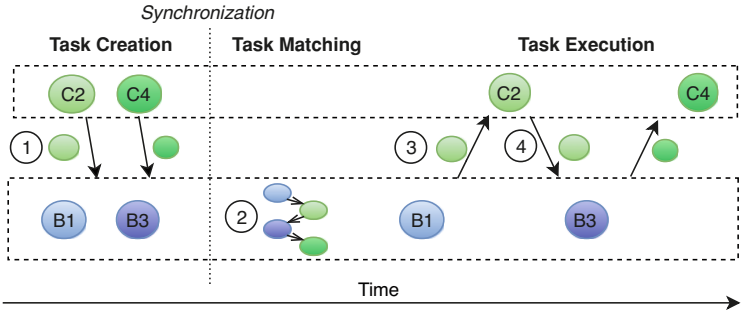


Fig. 2 Scheduler interaction required for handling remote task data dependencies

The discovery of the local task graphs thus happens in parallel and without immediate synchronization between the units, i.e., only the bold edges in Fig. 1 are immediately visible to the individual scheduler instances. In order to connect these trimmed task graphs, the schedulers need to exchange information on dependencies crossing its process boundary, i.e., dependencies referencing non-local global memory locations. The required interaction between the schedulers is depicted in Fig. 2. During the discovery of the trimmed local task graphs, schedulers communicate any encountered dependencies that reference non-local global memory to the unit owning the referenced memory ①. As soon as all dependencies have been communicated, the schedulers extend their local task graphs with the dependency information received from other units ②. A synchronization across all units is required to ensure that all relevant dependency information has been exchanged.

After the extension of the local task graphs, the units start the execution of the tasks. As soon as a task with a dependency to a task on a remote unit, e.g., through an input dependency previously communicated by the remote unit, has completed, the dependency release is communicated to the remote unit ③, where the task will eventually be executed. The completion of the execution is then communicated back to the first scheduler ④ to release any write-after-read dependency, e.g., the dependency $C_2 \rightarrow B_3$ in Fig. 1.

www.dbooks.org

2.1.2 Ordering of Dependencies

As described in Sect. 2.1.1, the local task graphs are discovered by each unit separately. Local edges in the local task graph are discovered similar to the matching rules of OpenMP, i.e., an input dependency refers to the previous output dependency referencing the same memory location (read-after-write), which in turn match with previous input and output dependencies on the same memory location (write-after-read and write-after-write).

However, since the local task graphs are discovered in parallel, the schedulers cannot infer any partial ordering of tasks and dependencies across process boundaries. More specifically, the blue scheduler in Fig. 1 cannot determine the relationship between the dependencies of tasks B_1, B_2, C_2, C_4. The schedulers thus have to rely on additional information provided by the user in the form of *phases* (as depicted in Fig. 1). A task and its output dependencies are assigned to the current phase upon their discovery. Input dependencies always refer to the last matching output dependency in any previous phase while output dependencies match with any previous local input dependency in the same or earlier phase and any remote input dependency in any earlier phase, up to and including the previous output dependency on the same memory location.

As an example, the input dependency of C_2 is assigned the phase $N + 1$ whereas the input dependency of C_4 is assigned the phase $N + 3$. This information can be used to match the output dependency of B_1 in phase N to the input dependency of C_2 and the output dependency of B_2 in phase $N + 2$ to the input dependency of C_4, creating the edges $B_1 \rightarrow C_2$ and $B_2 \rightarrow C_4$. The handling of write-after-read dependencies described in Sect. 2.1.1 creates the edge $C_2 \rightarrow B_2$. The handling of local dependencies happens independent of the phase.

In our model, conflicting remote dependencies in the same phase are erroneous as the scheduler is unable to reliably match the dependencies. Two dependencies are conflicting if at least one of them is non-local and at least one is an output dependency. This restriction allows the schedulers to detect synchronization errors such as underdefined phases and report them to the user. This is in contrast to the collective synchronization through barriers traditionally used in DASH, in which synchronization errors cannot be easily detected and often go unnoticed unless the resulting non-deterministic behavior leads to deviations in the application's results.

2.1.3 Implementation

A single task is created using the `async` function in DASH, which accepts both an action to be performed when the task is executed and a set of dependencies that describe the expected inputs and outputs of the task. In the example provided in Listing 1, every call to `async` (lines 5, 11, 19, and 27) is passed a C++ lambda in addition to input and output dependencies.

Instead of pure input dependencies, the example uses `copyin` dependencies, which combine an input dependency with the transfer of the remote memory range

into a local buffer. This allows for both a more precise expression of the algorithms and allows the scheduler to map the transfer onto two-sided MPI communication primitives, which may be beneficial on systems that do not efficiently support MPI RMA. The action performed by the task could still access any global memory location, keeping communication and synchronization decoupled in principle and retaining a one-sided programming model while allowing the use of two-sided communication in the background.

```
1  dash::Matrix<2, double> matrix{N, N, dash::TILE(NB), dash::TILE(NB)};
2
3  for (int k = 0; k < num_blocks; ++k) {
4    if (mat.block(k,k).is_local()) {
5      dash::tasks::async([&](){ potrf(matrix.block(k,k)); },
6        dash::tasks::out(mat.block(k,k)));
7    }
8    dash::tasks::async_fence(); // <- advance to next phase
9    for (int i = k+1; i < num_blocks; ++i)
10     if (mat.block(k,i).is_local())
11       dash::tasks::async([&](){
12           trsm(cache[k], matrix.block(k,i)); },
13         dash::tasks::copyin(mat.block(k,k), cache[k]),
14         dash::tasks::out(mat.block(k,i)));
15   dash::tasks::async_fence(); // <- advance to next phase
16   for (int i = k+1; i < num_blocks; ++i) {
17     for (int j = k+1; j < i; ++j) {
18       if (mat.block(j,i).is_local()) {
19         dash::tasks::async([&](){
     ↪
20             gemm(cache[i], cache[j], mat.block(j,i)); },
21           dash::tasks::copyin(mat.block(k,i), cache[i]),
22           dash::tasks::copyin(mat.block(k,j), cache[j]),
23           dash::tasks::out(mat.block(j,i)));
24       }
25     }
26     if (mat.block(i,i).is_local()) {
27       dash::tasks::async([&](){
28           syrk(cache[i], mat.block(i,i)); },
29         dash::tasks::copyin(mat.block(k,i), cache[i]),
30         dash::tasks::out(mat.block(i,i)));
31     }
32   }
33   dash::tasks::async_fence();  // <- advance to next phase
34 }
35 dash::tasks::complete(); // <- wait for all tasks to execute
```

Listing 1 Implementation of Blocked Cholesky Factorization using global task data dependencies in DASH. Some optimizations omitted for clarity

The specification of phases is done through calls to the `async_fence` function (lines 8, 15, and 33 in Listing 1). Similar to a barrier, it is the user's responsibility to ensure that all units advance phases in lock-step. However, the phase transition triggered by `async_fence` does not incur any communication. Instead, the call causes an increment of the phase counter, whose new value will be assigned to all ensuing tasks.

Eventually, the application waits for the completion of the execution of the global task graph in the call to `complete()`. Due to the required internal synchronization and the matching of remote task dependencies, the execution of all but the tasks in

www.dbooks.org

the first phase has to be post-poned until all its dependencies in the global task-graph are known. DASH, however, provides the option to trigger intermediate matching steps triggered by a phase increment and allows the specification of a an upper bound on the number of active phases[1] to avoid the need for discovering the full local task graph before execution starts. This way the worker threads executing threads may be kept busy while the main thread continues discovering the next window in the task graph.

In addition to the single task creation construct described above, DASH also provides the `taskloop()` construct, for which an example is provided in Listing 2. The `taskloop` function divides the iteration space `[begin, end)` into chunks that are assigned to tasks, which perform the provided action on the assigned subrange (lines 7–9). The user may control the size of each chunk (or the overall number of chunks to be created) by passing an instance of `chunk_size` (or `num_chunks`) to the call (Line 5). In addition, the call accepts a second lambda that is used to specify the dependencies of each task assigned a chunk (lines 11–14), which allows a depth-first scheduler to chain the execution of chunks of multiple data-dependent loops, effectively improving cache locality without changing the structure of the loops.

```
1  dash::Array<int> arr(N);
2
3  if (dash::myid() == 0) {
4    dash::tasks::taskloop(
5      arr.begin(), arr.end(), dash::tasks::chunk_size(10),
6      // task action
7      [&] (auto begin, auto end) {
8        // perform action on elements in [begin, end)
9      },
10     // generate out dependencies on elements in [begin, end)
11     [&] (auto begin, auto end, auto deps) {
12       for (auto it = begin; it != end; ++it)
13         *deps = dash::tasks::out(it);
14     });
15 }
```

Listing 2 Example of using the `dash::taskloop` in combination with a dependency generator

2.1.4 Results: Blocked Cholesky Factorization

The implementation of the Blocked Cholesky Factorization discussed in Sect. 2.1.3 has been compared against two implementations in PaRSEC. The first implementation uses the parameterized task graph (PTG), in which the problem is described as an directed acyclic graph in a domain-specific language called JDF. In this version,

[1] A phase is considered active while a task discovered in that phase has not completed execution.

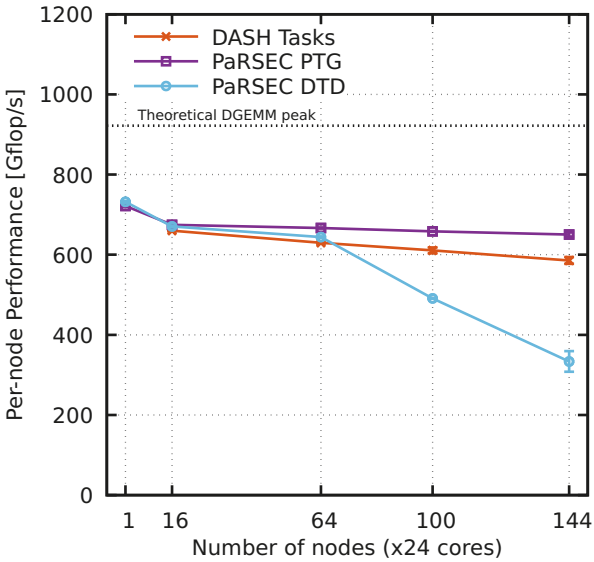


Fig. 3 Per-node weak-scaling performance of Blocked Cholesky Factorization of a matrix with leading dimension $N = 25k/node$ and block size $NB = 320$ on a Cray XC40 (higher is better)

the task graph is not dynamically discovered but is instead inherently contained within the resulting binary.

The second PaRSEC version uses the Dynamic Task Discovery (DTD) interface of PaRSEC, in which problems are expressed with a global view, i.e., all processes discover the global task graph to discover the dependencies to tasks executing on remote processes.

For all runs, a background communication thread has been employed, each time running on a dedicated core, leading to one main thread and 22 worker threads executing the application tasks on the Cray XC40.

The results presented in Fig. 3 indicate that PaRSEC PTG outperforms both DTD and DASH, due to the missing discovery of tasks and their dependencies. DTD exhibits a drop in per-node performance above 64 nodes, which may be explained with the global task graph discovery. Although the per-node performance of DASH does not exhibit perfect scaling, it still achieves about 80% of the performance of PaRSEC PTG at 144 nodes.

2.1.5 Related Work

HPX [18] is an implementation of the ParalleX [17] programming paradigm, in which tasks are spawned dynamically and moved to the data, instead of the data being moved to where the task is being executed. HPX is locality-agnostic in that distributed parallelism capabilities are implicit in the programming model, rather than explicitly exposed to the user. An Active Global Address Space (AGAS) is used to transparently manage the locality of global objects. Synchronization of tasks

www.dbooks.org

is expressed using futures and continuations, which are also used to exchange data between tasks.

In the Active Partitioned Global Address Space (APGAS) [27], in contrast, the locality of so-called places is explicitly exposed to the user, who is responsible for selecting the place at which a task is to be executed. Implementations of the APGAS model can be found in the X10 [9] and Chapel [8] languages as well as part of UPC and UPC++ [21].

The Charm++ programming system encapsulates computation in objects that can communicate using message objects and can be migrated between localities to achieve load balancing.

Several approaches have been proposed to facilitate dynamic synchronization by waiting for events to occur. AsyncShmem is an extension of the OpenShmem standard, which allows dynamic synchronization of tasks across process boundaries by blocking tasks waiting for a state change in the global address space [14]. The concept of phasers has been introduced into the X10 language to implement non-blocking barrier-like synchronization, with the distinction of readers and writers contributing to the phaser [29].

Tasklets have recently been introduced to the XcalableMP programming model [35]. The synchronization is modeled to resemble message-based communication, using data dependencies for tasks on the same location and notify-wait with explicitly specified target and tags.

Regent is a region- and task-based programming language that is compiled into C++ code using the Legion programming model to automatically partition the computation into logical regions [4, 30].

The PaRSEC programming system uses a domain specific language called JDF to express computational problems in the form of a parameterized task graph (PTG) [7]. The PTG is implicitly contained in the application and not discovered dynamically at runtime. In contrast to that, the dynamic task discovery (DTD) frontend of PaRSEC dynamically discovers the global task-graph, i.e., each process is aware of all nodes and edges in the graph.

A similar approach is taken by the sequential task flow (STF) frontend of StarPU, which complements the explicit MPI send/recv tasks to encapsulate communication in tasks and implicitly express dependencies across process boundaries [1].

Several task-based parallelization models have been proposed for shared memory concurrency, including OpenMP [3, 24], Intel thread building blocks (TBB) [25] and Cilk++ [26] as well as SuperGlue [34]. With ClusterSs, an approach has been made to introduce the APGAS model into OmpSs [32].

2.2 Dynamic Hardware Topology

Portable applications for heterogeneous hosts adapt communication schemes and virtual process topologies depending on system components and the algorithm scenario. This involves concepts of vertical and horizontal locality that are not based

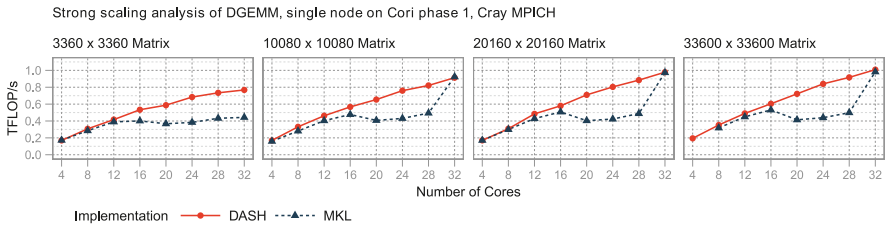


Fig. 4 Strong scaling of matrix multiplication on single node for 4 to 32 cores with increasing matrix size N × N on Cori phase 1, Cray MPICH

on latency and throughput as distance measure. For example in a typical accelerator-offloading scenario, data distribution to processes optimizes for horizontal locality to reduce communication distance between collaborating tasks. For communication in the reduction phase, distance is measured based on vertical locality.

The goal to provide a domain-agnostic replacement for the C++ Standard Template Library (STL) implies portability as a crucial criterion for every model and implementation of the DASH library. This includes additional programming abstractions provided in DASH, such as n-dimensional containers which are commonly used in HPC. These are not part of the C++ standard specifications but comply with its concepts. Achieving portable efficiency of PGAS algorithms and containers that satisfy semantics of their conventional counterparts is a multivariate, hard problem, even for the seemingly most simple use cases.

Performance evaluation of the of the DASH NArray and dense matrix-matrix multiplication abstractions on different system configurations substantiated the portable efficiency of DASH. The comparison also revealed drastic performance variance of the established solutions, for example node-local DGEMM of Intel MKL on Cori phase 1 shown in Fig. 4 which apparently expected a power of two amount of processing cores for multi-threaded scenarios.

The DASH variant of DGEMM internally uses the identical Intel MKL distribution for multiplication of partitioned matrix blocks but still achieves robust scaling. This is because DASH implements a custom, adaptive variant of the SUMMA algorithm for matrix-matrix multiplication and assigns one process per core, each using MKL in sequential mode. This finding motivated to find abstractions that allow expressions for domain decomposition and process placement depending on machine component topology. In this case: to group processes by NUMA domains with one process per physical core.

2.2.1 Locality-Aware Virtual Process Topology

In the DASH execution model, individual computation entities are called *units*. In the MPI-based implementation of the DASH runtime, a unit corresponds to an MPI rank but may occupy a locality domain containing several CPU cores or, in principle, multiple compute nodes.

www.dbooks.org

Units are organized in hierarchical *teams* to match the logical structure of algorithms and machine components. Each unit is an immediate member of exactly one team at any time, initially in the predefined team `ALL`. Units in a team can be partitioned into child teams using the team's `split` operation which also supports locality-aware parameters.

On systems with asymmetric or deep memory hierarchies, it is highly desirable to split a team such that locality of units within every child team is optimized. A locality-aware split at node level could group units by affinity to the same NUMA domain, for example. For this, locality discovery has been added to the DASH runtime. Local hardware information from hwloc, PAPI, libnuma, and LIKWID of all nodes is collected into a global, uniform data structure that allows to query locality information by process ID or scope in the memory hierarchy.

This query interface proved to be useful for static load balancing on heterogeneous systems where team are split depending on the machine component capacities and capabilities. These are stored in a hierarchy of domains with two *property maps*:

Capabilities invariant hardware locality properties that do not depend on the locality graph's structure, like the number of threads per core, cache sizes, or SIMD width

Capacities derivative properties that might become invalid when the graph structure is modified, like memory in a NUMA domain available per unit

Figure 5 outlines the data structure and its concept of hardware abstraction in a simplified example of a topology-aware split. Domain capacities are accumulated from its subdomains and recalculated on restructuring. Team 1 and 2 both contain twelve cores but a different number of units. A specific unit's maximum number of threads is determined by the number of cores assigned to the unit and the number of threads per core.

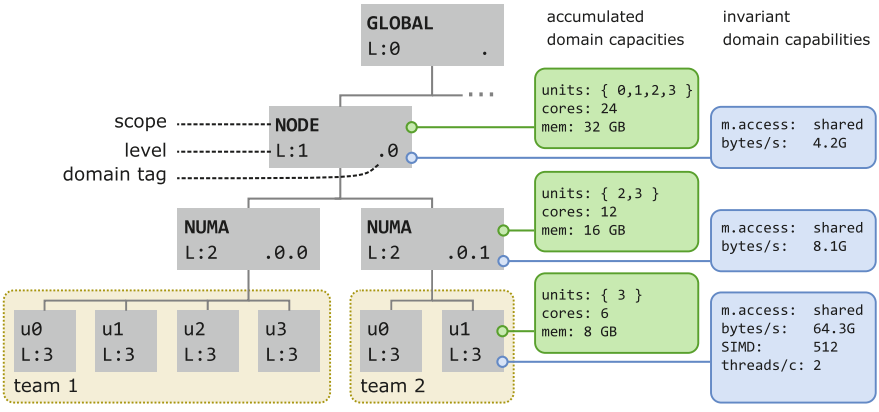


Fig. 5 Domains in a locality hierarchy with domain attributes in dynamically accumulated capacities and invariant capabilities

2.2.2 Locality Domain Graph

The machine component topology of the DASH runtime to support queries and topology-aware restructuring extends the tree-based hwloc topology model to represent properties and relations of machine components in a graph structure. It evolved to the *Locality Domain Graph* (LDG) concept which is available as the standalone library *dyloc*.[2]

In formal terms, a locality domain graph models hardware topology as directed, multi-indexed multigraph. In this, nodes represent *Locality Domains* that refer to any physical or logical component of a distributed system with memory and computation capabilities, corresponding to *places* in X10 or Chapel's *locales* [8]. Edges in the graph are directed and denote the following relationships, for example:

- **Containment** indicating that the target domain is logically or physically contained in the source domain
- **Alias** source and target domains are only logically separated and refer to the same physical domain; this is relevant when searching for a shortest path, for example
- **Leader** the source domain is restricted to communication with the target domain

2.2.3 Dynamic Hardware Locality

Dynamic locality support requires means to specify transformations on the physical topology graph as *views*. Views realize a projection but must not actually modify the original graph data. Invariant properties are therefore stored separately and assigned to domains by reference only.

Conceptually, multi-index graph algebra can express any operation on a locality domain graph, but complex to formulate. When a topology is projected to an acyclic hierarchy, transformations like partitioning, selection and grouping of domains can be expressed in conventional relational or set semantics. A partition or contraction of a topology graph can be projected to a tree data structure and converted to a hwloc topology object (Fig. 6).

A locality domain topology is specific to a team and only contains domains that are populated by the team's units. At initialization, the runtime initializes the default team `ALL` as root of the team hierarchy with all units and associates the team with the global locality graph containing all domains of the machine topology. When a team is split, its locality graph is partitioned among child teams such that a single partition is coherent and only contains domains with at least one leaf occupied by a unit in the child team.

In a map-reduce scenario, dynamic views on machine topology to express for domain decomposition and process placement depending on machine component

[2] https://github.com/dash-project/dyloc.

www.dbooks.org

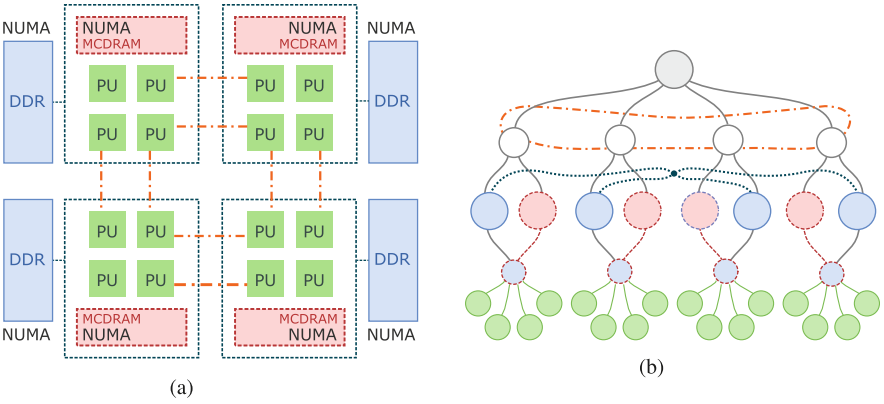


Fig. 6 Illustration of a hardware locality domain graph as a model of node-level system architectures that cannot be correctly or unambiguously represented in a single tree structure. (**a**) Cluster in Intel Knights Landing, configured in Sub-NUMA clustering, Hybrid mode. Contains a quarter of the processor's cores, MCDRAM local memory, affine to DDR NUMA domain. (**b**) Exemplary graph representation of Knights Landing topology in (**a**). Vertex categories model different aspects of component relationships, like cache-coherence and adjacency

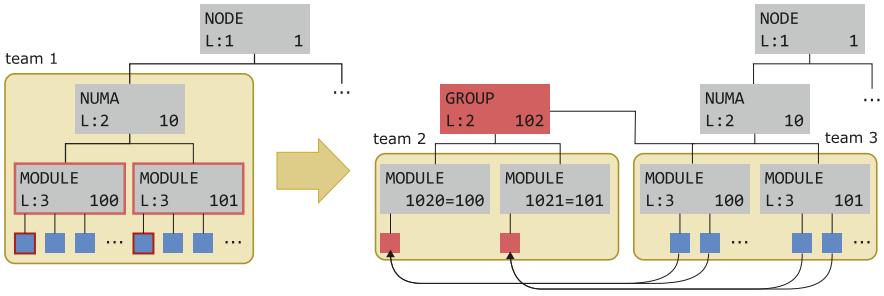


Fig. 7 Illustration of the domain grouping algorithm to define a leader group for vertical communication. One core is selected as leader in domains 100 and 110 and separated into a group. To preserve the original topology structure, the group includes their parent domains and is added as a subdomain of their lowest common ancestor

topology and improve portable efficiency. In the map phase, the algorithm is mostly concerned with *horizontal locality* in domain decomposition to distribute data according to the physical arrangement of cooperating processes. In the reduce phase, *vertical locality* of processes in the component topology determines efficient upwards communication of partial results. The locality domain graph can be used to project hardware topology to tree views for both cases. Figure 7 illustrates a locality-aware split of units in two modules such that one unit per module is selected for upwards communication. This principle is known as *leader communication scheme*. Partial results of units are then first reduced at the unit in the respective leader team. This drastically reduces communication overhead as the physical bus between the

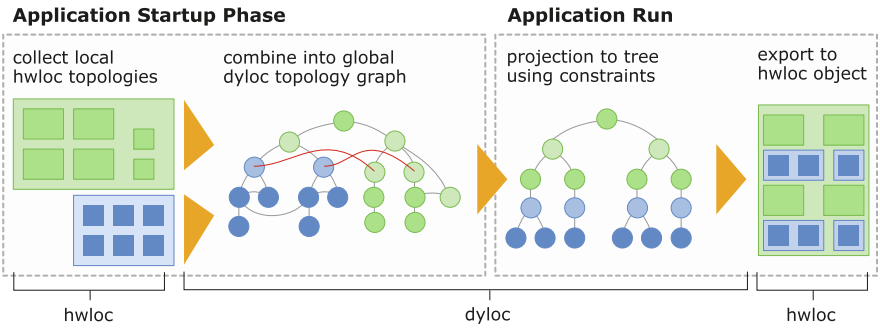


Fig. 8 Using dyloc as intermediate process in locality discovery

modules and their NUMA node is only shared by two leader processes instead of all processes in the modules (Fig. 8).

2.2.4 Supporting Portable Efficiency

As an example of both increased depth of the machine hierarchy and heterogeneous node-level architecture, the SuperMIC system[3] consists of 32 compute nodes with symmetric hardware configuration of two NUMA domains, each containing an Ivy Bridge (8 cores) host processor and a Xeon Phi "Knights Corner" coprocessors (Intel MIC 5110P) as illustrated in Fig. 9.

For portable work load balancing on heterogeneous systems, domain decomposition and virtual process topology must dynamically adapt the machine components' inter-connectivity, capacities and capabilities.

Capacities: Total memory capacity on MIC modules is 8 GB for 60 cores, significantly less than 64 GB for 32 cores on host level

Capabilities: MIC cores have a base clock frequency of 1.1 GHz and 4 SMT threads, with 2.8 GHz and 2 SMT threads on host level

To illustrate the benefit of dynamic locality, we briefly discuss the implementation of the `min_element` algorithm in DASH. Its original variant is implemented as follows: domain decomposition divides the element range into contiguous blocks of identical size. All units then run a thread-parallel scan on their local block for a local minimum and enter a collective barrier once it has been found. Once all units finished their local work load, local results are reduced to the global minimum.

Listing 3 contains the abbreviated implementation of the `min_element` scenario utilizing runtime support based on a dynamic hardware locality graph.

[3] https://www.lrz.de/services/compute/supermuc/supermic.

www.dbooks.org

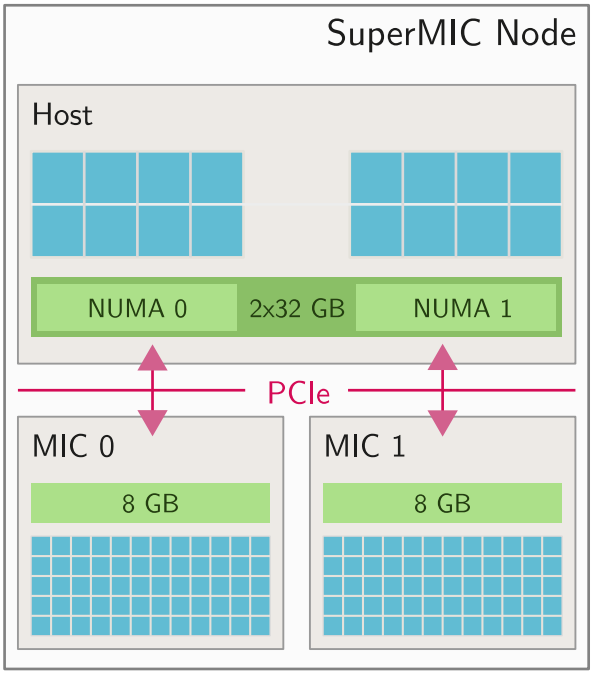


Fig. 9 SuperMIC node

Dynamic topology queries are utilized in three essential ways to improve overall load-balance: In domain decomposition (lines 3–6), to determine the number of threads available to the respective unit (line 19) and for a simple leader-based communication scheme (lines 8–10, 26).

This implementation achieves portable efficiency across systems with different memory hierarchies and hardware component properties, and dynamically adapts to runtime-specific team size, range size, and available hardware components assigned to the team. Figure 10 shows timeline plots comparing time to completion and process idle time from a benchmark run executed on SuperMIC.

```
1  // Dynamic topology-aware domain decomposition depending on
2  // machine component properties and number of units in team:
3  TeamLocality       tloc(dash::Team::All());
4  LocBalancedPattern pattern(array_size, tloc);
5  dash::Array<T>     array(pattern);
6
7  GlobIt min_element(GlobIt first, GlobIt last) {
8    auto uloc     = UnitLocality(myid());
9    auto leader   = uloc.at_scope(scope::MODULE)
10                       .unit_ids()[0];
11   auto loc_min  = first;
12
13   // Allocate shared variable for reduction result at leader:
14   dash::Shared<GlobIt> glob_min(leader);
15   // Allocate shared array for local minimum values:
```

```
16   dash::Array<GlobIt>(dash::Team::All().size()) loc_mins;
17
18   // Dynamic query of locality runtime for number of threads:
19   auto nthreads = uloc.num_threads();
20   #pragma omp parallel for num_threads(nthreads)
21   for (...) { /* ... find local result ... */ }
22   // Local write, no communication
23   loc_mins[my_id] = loc_min;
24   dash::barrier();
25
26   if (myid() == leader) {
27      // leader reduces local results (instead of all-to-all
28      // reduction)
29      glob_min = std::min_element(loc_mins.begin(),
30                                  loc_mins.end());
31
32   }
33   // ...
34 }
```

Listing 3 Code excerpt of the modified min_element algorithm

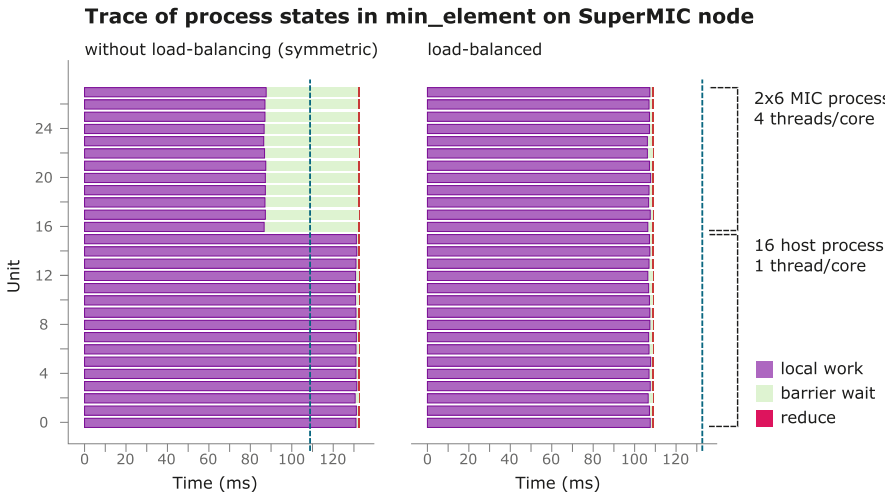


Fig. 10 Trace of process activities in the `min_element` algorithm exposing the effect of load balancing based on dynamic hardware locality

www.dbooks.org

3 DASH C++ Data Structures and Algorithms

The core of DASH is formed by data structures and algorithms implemented as C++ templates. These components are conceptually modeled after their equivalents in the C++ standard template library, shortening learning curves and increasing programmer productivity. A basic DASH data structure is the distributed array with configurable data distribution (`dash::Array`) which closely follows the functionality of a STL `vector` except for the lack of runtime resizing. DASH also offers a multidimensional array and supports a rich variety of data distribution patterns [11]. A focus of the second half of the funding period was placed on *smart* data structures which are more specialized and support users in the development of certain types of applications. One such data structures for the development of stencil-based applications is described in Sect. 3.1.

Similar to data structures, DASH also offers generalized parallel algorithms. Many of the over 100 generic algorithms contained in the STL have an equivalent in DASH (e.g., `dash::fill`). One of the most useful but also challenging algorithms is sorting and Sect. 3.2 describes our implementation of scalable distributed sorting in the DASH library.

3.1 Smart Data Structures: Halo

Typical data structure used in ODE/PDE solvers or 2D/3D image analyzers are multi-dimensional arrays. The DASH NArray distributes data elements of a structured data grid and can be used similar to STL containers. But PDE solvers use stencil operations, not using the current data elements (center), but surrounding data elements (neighbors) as well. The use of the NArray it self is highly inefficient with stencil operations, because neighbors located in another sub-arrays may require remote access (via RDMA or otherwise). A more efficient approach is the use of so called “halo areas”. These areas contain copies of all required neighbor elements located on other compute nodes. The halo area width depends on the shape of the stencils and is determined by the largest distance from the center (per dimension). The stencil shape defines all participating data elements—center and neighbors. Figure 11 shows two 9-point stencils with different shapes. The first stencil shape (a) accesses ± 2 data elements in both horizontal and vertical direction and the second one (b) accesses ± 1 stencil point in each direction. While the stencil shape Fig. 11a needs four halo areas with a width of two data elements. The other stencil shape requires eight halo areas with a width one data elements. Using halo areas ensures local data access for all stencil operations used on each sub-array.

The Dash NArray Halo Wrapper wraps the local part of the NArray and automatically sets up a halo environment for stencil codes and halo accesses. Figure 12 shows an overview about all main components, which are explained in the following.

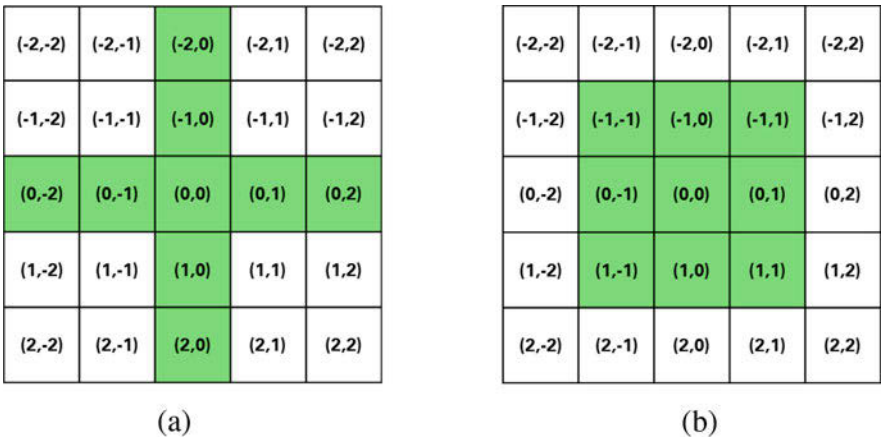


Fig. 11 Two shapes of a 9-point stencil. (**a**) ± 2 center stencil in horizontal and vertical directions. (**b**) Center ± 1 stencil point in each direction

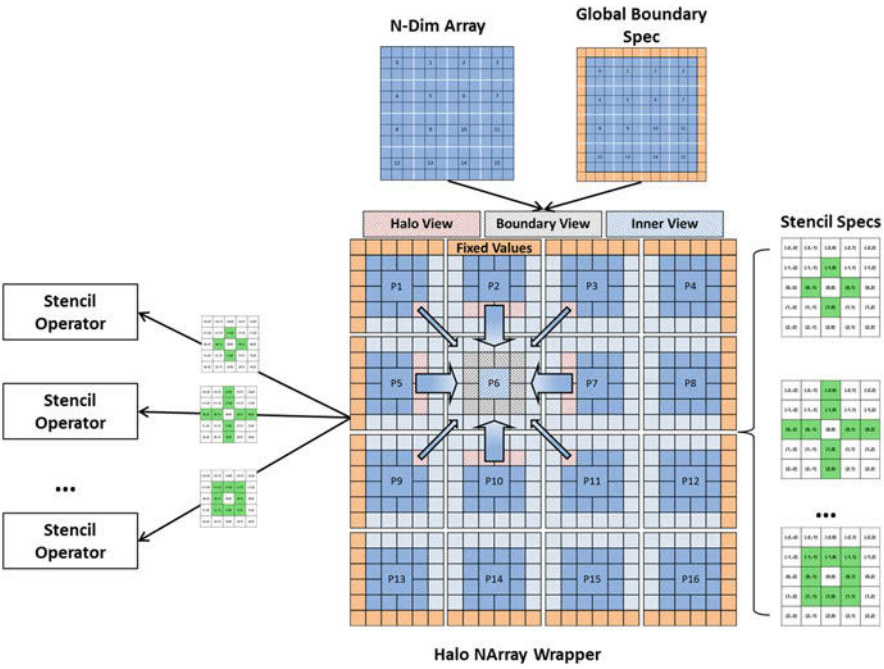


Fig. 12 Architecture of the DASH Halo NArray Wrapper

www.dbooks.org

3.1.1 Stencil Specification

The discretization of the problem to be solved always determines the stencil shape to be applied on the structured grid. For a DASH based code this has to be specified as a stencil specification (StencilSpec), which is a collection of stencil points (StencilPoint). A StencilPoint consists of coordinates relative to the center and an optional weight (coefficient). The StencilSpec specified in Listing 4 describes the stencil shown in Fig. 11b. The center doesn't have to be declared directly.

```
1 using PointT = dash::halo::StencilPoint<2>;
2 dash::halo::StencilSpec<PointT,6> stencil_spec(
3               PointT(-1,-1), PointT(-1, 0), PointT(-1,1),
4               PointT( 0,-1),                , PointT( 0,1),
5               PointT( 1,-1), PointT( 1, 0), PointT( 1,1));
```

Listing 4 Stencil specification for an 9-point stencil

3.1.2 Region and Halo Specifications

The region specification (RegionSpec) defines the location of all neighboring partitions. Every unit keeps 3^n regions representing neighbor partitions for "left", "middle", and "right" in each of the n dimensions. All regions are identified by a region index and a corresponding region coordinate. Indexing are done with the Row Major linearization (last index grows fastest). Figure 13 shows all possible regions with its indexes and coordinates for a two dimensional scenario. Region 4 with the coordinates (1,1) is mapped to the center region and represents the local partition. Region 6 (2,0) points to a remote partition located in the south west.

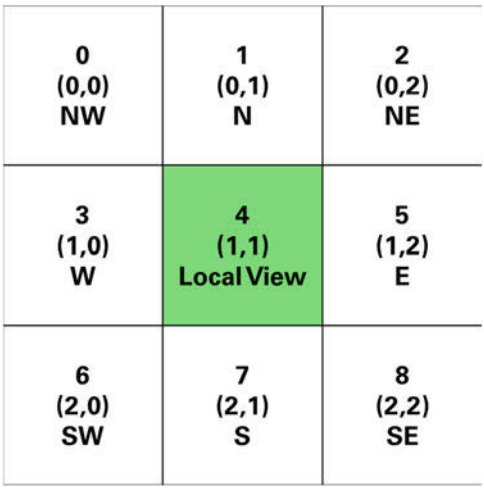


Fig. 13 Mapping of region coordinates and indexes

The halo specification (HaloSpec) uses the RegionSpec to map neighbor partitions which are the origins for halo copies to the local halo areas. From one or multiple StencilSpecs it infers which neighbor partitions are necessary. In case no StencilPoint has a negative offset from the center in horizontal direction, no halo regions for the 'NW', 'W', and 'SW' (Fig. 13) need to be created. If no StencilPoint has diagonal offsets (i.e. only one non-zero coordinate in the offsets) the diagonal regions 'NW','NE', 'SW', and 'SE' can be omitted.

3.1.3 Global Boundary Specification

Additionally, the global boundary specification (GlobalBoundarySpec) allows to control the behavior at the outside of the global grid. For convenience, three different scenarios are supported. The default setting is `NONE` meaning that there are no halo areas in this direction. Therefore, the stencil operations are not applied in the respective boundary region where the stencil would require the halo to be present. As an alternative, the setting `CYCLIC` can be set. This will wrap around the simulation grid, so that logically the minimum coordinate becomes a neighbor to the maximum coordinate. Furthermore, the setting `CUSTOM` creates a halo area but never performs automatic update of its elements from any neighbors. Instead, this special halo area can be written by the simulation (initially only or updated regularly). This offers a convenient way to provide boundary conditions to a PDE solver. The GlobalBoundarySpec can be defined separately per dimension.

3.1.4 Halo Wrapper

Finally, using the aforementioned specifications as inputs, the halo wrapper (HaloWrapper) creates HaloBlocks for all local sub-arrays. The halo area extension is derived from all declared StencilSpecs by determining the maximum offset of any StencilPoint in the given direction.

The mapping between the local HaloBlocks and the halo regions pointing to the remote neighbor data elements is subjected to the HaloWrapper, as well as the orchestration of efficient data transfers for halo data element updates. The data transfer has to be done block-wise instead of element-wise to gain decent performance. While the HaloBlock can access contiguous memory, the part of the neighbor partition marked as halo area, usually can't be accessed contiguously—compare Fig. 14. Therefore, the HaloWrapper relies on DART's support for efficient strided data transfers.

The halo data exchange can be done per region or for all regions at once. It can be called asynchronously and operates independent between all processes and doesn't use process synchronization. The required subsequent wait operation waits for local completion only.

www.dbooks.org

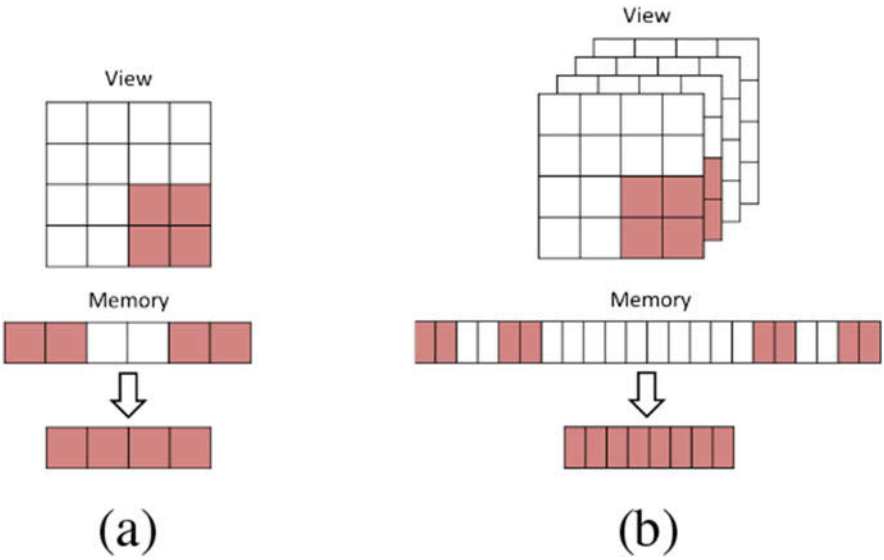


Fig. 14 Halo data exchange of remote strided data to contiguous memory: (**a**) in 2D a corner halo region has a fixed stride, whereas (**b**) in 3D the corner halo region has two different strides

3.1.5 Stencil Operator and Stencil Iterator

So far, the HaloWrapper was used to create HaloBlocks to fit all given StencilSpecs. Besides that, the HaloWrapper also provides specific views and operations for each StencilSpec.

First, for every StencilSpec the HaloWrapper provides a StencilOperator with adapted *inner* and *boundary* views. The inner view contains all data elements that don't need a halo area when using the given stencil operation. All other data elements are marked via the boundary view. These two kind of views are necessary to overlap the halo data transfer with the inner computation. The individual view per StencilSpec allows to make the inner view as large as possible, regardless of other StencilSpecs.

Second, the HaloWrapper offers StencilSpec specific StencilIterators. They iterate over all elements assigned by a given view (inner) or a set of views (boundary). With these iterators center elements can be accessed—equivalent to STL iterators—via the dereference operator. Neighboring data elements can be accessed with a provided method. Stencil points pointing to elements within a halo area, are resolved automatically without conditionals in the surrounding code. StencilIterators can be used with random access, but are optimized for the increment operator.

3.1.6 Performance Comparison

A code abstraction hiding complexity is useful only, if no or minor performance impact is added. Therefore, a plain MPI implementation of a heat equation was compared to a DASH based one regarding weak and strong scaling behavior. All measurements were performed on the Bull HPC-Cluster "Taurus" at ZIH, TU Dresden. Each compute node has two Haswell E5-2680 v3 CPUs at 2.50 GHz with 12 physical cores each and 64 GB memory. Both implementations were built with gcc 7.1.0 and OpenMPI 3.0.0.

The weak scaling scenario increases the number of grid elements proportional to the number of compute nodes. The accumulated main memory is almost entirely used up by each compute grid. Figure 15 shows that both implementations almost have identical and perfect weak scaling behavior. Note that the accumulated waiting times differ significantly. This is due to two effects. One is contiguous halo areas (north and south) vs. strided halo areas (east and west). The other is intra node vs. inter node communication.

The strong scaling scenario uses $55{,}000^2$ grid elements to fit into the main memory of a single compute node. It is solved with 1 to 768 *CPU cores* (== MPI ranks), where 24 cores equals to one full compute node and 768 cores to 32 compute nodes. Figure 16 shows again an almost identical performance behavior between DASH and MPI for the total runtime. Notably, both show the same performance artifact around 16 to 24 cores. This can be ascribed to an increased number of last level cache load misses which indicates that both implementations are memory bound at this number of cores per node.

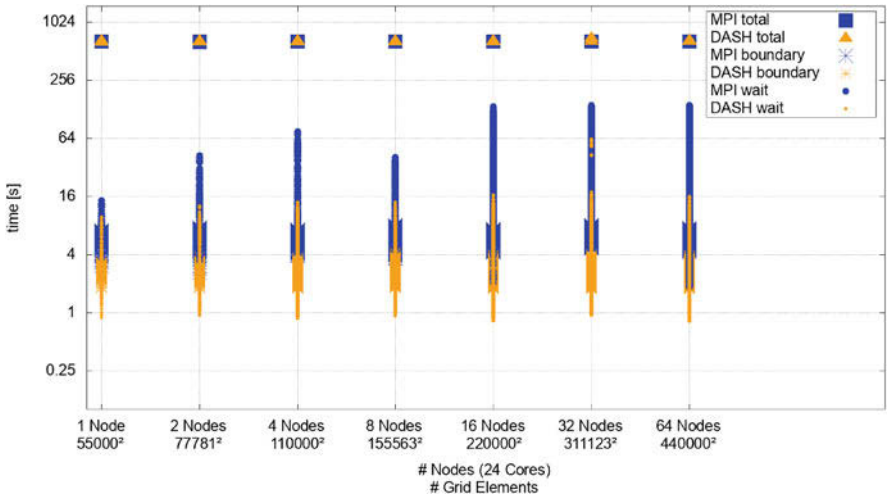


Fig. 15 Weak scaling in DASH vs. MPI

www.dbooks.org

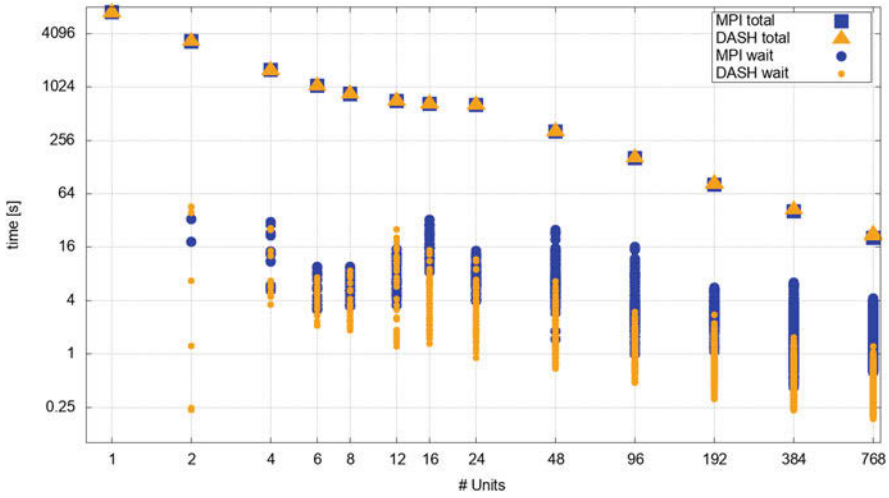


Fig. 16 Strong scaling for 55,000 × 55,000 elements in DASH vs. MPI. Effective wait time for asynchronous halo exchanges is shown in addition

3.2 Parallel Algorithms: Sort

Sorting is one of the most important and well studied non-numerical algorithms in computer science and serves as a basic building block in a wide spectrum of applications. A notable example in the scientific domain are N-Body particle simulations which are inherently communication bound due to load imbalance. Common strategies to mitigate this problem include redistributing particles according to a space filling curve (e.g., Morton Order) which can be achieved with sorting. Other interesting use cases which can be addressed using DASH are Big Data applications, e.g., Google PageRank.

Key to achieve performance is obviously to minimize communication. This applies not only to distributed memory machines but to shared memory architectures as well. Current supercomputers facilitate nodes with large memory hierarchies organized in a growing number of NUMA domains. Depending on the data distribution, sorting is subject to a high fraction of data movement and the more we communicate across NUMA boundaries the more negative the result performance impact becomes.

In the remainder of this section we briefly describe the problem of sorting in a more formal manner and summarize the basic approaches in related work. It follows a more detailed elaboration of our sorting algorithm [20]. Case studies on both distributed and shared memory demonstrate our performance efficiency. Results reveal that we can outperform state of the art implementations with our PGAS algorithm.

3.2.1 Preliminaries

Let X be a set of N keys evenly partitioned among P processors, thus, each processor contributes $n_i \sim N/P$ keys. We further assume there are no duplicate keys which can technically be achieved in a straightforward manner. Sorting permutes all keys by a predicate which is a binary relation in set X. Recursively applying this predicate to any ordered pair (x, y) drawn from X enables to determine the rank of an element $I(x) = k$ with x as the k-th order statistic in X. Assuming our predicate is *less than* (i.e., $<$) the output invariant after sorting guarantees that for any two subsequent elements $x, y \in X$

$$x < y \Leftrightarrow I(x) < I(y).$$

Scientific applications usually require a balanced load to maximize performance. Given a load balance threshold ϵ, *local balancing* means that in the sorted sequence each processor P_i owns at most $N(1 + \epsilon)/P$ keys. This does not always result in a *globally balanced* load which is an even stronger guarantee.

Definition 1 For all $i \in \{1..P\}$ we have to determine splitter S_i to partition the input sequence into P subsequences such that

$$\frac{Ni}{P} - \frac{N\epsilon}{2P} \leq I(s_i) \leq \frac{Ni}{P} + \frac{N\epsilon}{2P}$$

Determining these splitters boils down to the *k-way selection* problem which is a core algorithm in this work. If $\epsilon = 0$ we need to perfectly partition the input which increases communication complexity. However, it often is the easiest solution in terms of programming productivity which is a major goal of the DASH library.

3.2.2 Related Work

Sorting large inputs can be achieved through parallel sample sort which is a generalization of *Quicksort* with multiple pivots [5]. Each processor partitions local elements into p pieces which are obtained out of a sufficiently large sample of the input. Then, all processors exchange elements among each other such that piece i is copied to processor i. In a final step, all processors sort received pieces locally, resulting in a globally sorted sequence. *Perfect partitioning* can be difficult to achieve as splitter selection is based only on a sample of the input.

In parallel p-way mergesort each processor first sorts the local data portion and subsequently partitions it, similar to sample sort, into p pieces. Using an ALL-TO-ALL exchange all pieces are copied to the destination processors which finally merge them to obtain a globally sorted sequence. Although this algorithm has worse isoefficiency due to the partitioning overhead compared to sample sort, *perfect partitioning* becomes feasible since data is locally sorted.

www.dbooks.org

Scalable sorting algorithms are compromises between these two extremes and apply various strategies to mitigate negative performance impacts of splitter selection (partitioning) and ALL-TO-ALL communication [2, 15, 16, 31]. Instead of communicating data only once, partitioning is done recursively from a coarse-grained to a more fine-grained solution. Each recursion leads to independent subpartitions until the solution is found. Ideally, the level of recursion maps to the underlying hardware resources and network topology.

This work presents two contributions. First, we generalize a distributed selection algorithm to achieve scalable partitioning [28]. Second, we address the problem of communication-computation overlap in the ALL-TO-ALL exchange, which is conceptually limited in MPI as the underlying communication substrate.

3.2.3 Histogram Sort

The presented sorting algorithm consists of four supersteps as delineated in Fig. 17.

Local Sort Sorts the local portion using a fast shared memory algorithm.
Splitting Each processor partitions the local array into p pieces. We generalize distributed selection to a p-way multiselect.
Data Exchange Each processor exchanges piece i with processor i according to the splitter boundaries.
Local Merge Each processor merges the received sorted pieces.

Splitting is based on distributed selection [28]. Instead of finding one pivot we collect multiple pivots (splitters) in a single iteration, one for each *active* range. If a pivot matches a specific rank we do not consider this range anymore and discard it from the set of active ranges. Otherwise, we examine each of the two resulting

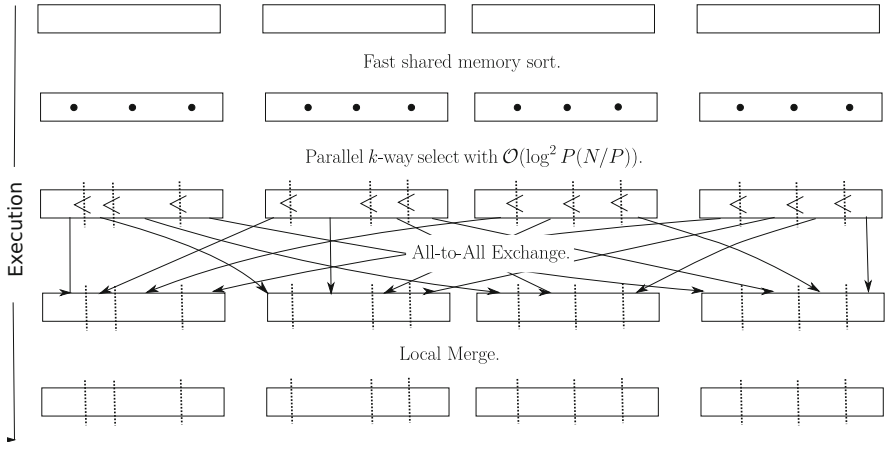


Fig. 17 Algorithmic schema of `dash::sort` with four processors ($P = 4$)

subranges whether they need to be considered in future iterations and add them to the set of active ranges. The vector of splitters follows from Definition 1 (page 126) and is the result of a prefix sum over the local capacities of all processors.

Another difference compared to the original algorithm [28] is that in our case we replace the local partition with binary search which is possible due to our initial sorting step in all processors. Thus, to determine a local histogram over p pieces requires logarithmic computation complexity instead of linear complexity. A global histogram to determine if all splitters are valid (or if we need to refine the boundaries) requires a single REDUCE over all processors with logarithmic communication complexity.

The question how many iterations we need is answered as follows. We know that for the base case with only two processors (i.e., only one splitter) distributed selection has a recursive depth of $O(\log p)$. This follows from the weighted median for the pivot selection which guarantees a reduction of the working set by at least one quarter each iteration. As described earlier instead of a single pivot we collect multiple pivots in a single iteration which we achieve by a list of active ranges. Although the local computation complexity increases by a factor of $O(\log p)$ the recursion depth does not change. Hence, the overall communication complexity is $O(\log^2 p)$ including the REDUCE call each iteration.

After successfully determining the splitters all processors communicate the locally partitioned pieces with an ALL-TO-ALL exchange. Merging all received pieces leads to a globally sorted sequence over all processors. Due to the high communication volume communication-computation overlap is required to achieve good scaling efficiency. However, for collective operations MPI provides a very limited interface. While we can use a non-blocking ALL-TO-ALL we cannot operate on partially received pieces. For this reason we designed our own ALL-TO-ALL algorithm to pipeline communication and merging. Similar to the Butterfly algorithm processor i sends to destination $(i + r) \pmod p$ and receives from $(i - r) \pmod p$ in round r [33]. However we schedule communication requests only as long as a communication buffer of a fixed length is not completely allocated. As soon as *some* communication requests complete we schedule new requests while merging the received chunks. PGAS provides additional optimizations. For communication within a shared memory node we use a cache-efficient ALL-TO-ALL algorithm to minimize negative cache effects among the involved processors. Instead of scheduling send receive pairs processor ranks are reordered according to a Morton order. Data transfer is performed using one-sided communication mechanisms. Similar optimizations can be applied to processor pairs running on nearby nodes. We are preparing a paper to describe the involved optimizations in more detail. First experimental evaluations reveal that we can achieve up to 22% speedup compared to a single ALL-TO-ALL followed by a p-way local merge.

In the next section we demonstrate our performance scalability against a state-of-the-art implementation in Charm++.

www.dbooks.org

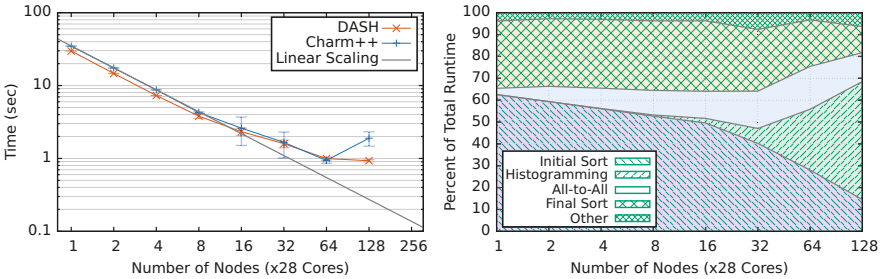


Fig. 18 Strong scaling study with Charm++ and DASH. (**a**) Median execution time. (**b**) Strong scaling behavior of `dash::sort`

3.2.4 Evaluation and Conclusion

We conducted the experiments on SuperMUC Phase 2 hosted at the Leibnitz Supercomputing Center. This system is an island-based computing cluster, each equipped with 512 nodes. Each node has two Intel Xeon E5-2697v3 14-core processors with a nominal frequency of 2.6 GHZ and 64 GB of memory, although only 56 GB are usable due to the operating system. Computation nodes are interconnected in a non-blocking fat tree with Infiniband FDR14 which achieves a peak bisection bandwidth of 5.1 TB/s. We compiled our binaries with Intel ICC 18.0.2 and linked the Intel MPI 2018.2 library for communication. The Charm++ implementation was executed using the most recent stable release.[4] On each node we scheduled only 16 MPI ranks (28 cores available) because the Charm++ implementation requires the number of ranks to be a power of two. We emphasize that our implementation in DASH does not rely on such constraints.

The strong scaling performance results are depicted in Fig. 18a. We sort 28 GBytes of uniformly distributed 64-bit signed integers. This is the maximum memory capacity on a single node because our algorithm is not in-place. We always report the median time out of 10 executions along with the 95% confidence interval, excluding an initial warmup run. For Charm++ we can see wider confidence intervals. We attribute this to a volatile histogramming phase which we can see after analyzing generated log files in the Charm++ experiments. Overall, we observe that both implementations achieve nearly linear speedup with a low number of cores. Starting from 32–64 nodes scalability gets worse. DASH still achieves a scaling efficiency of $\approx$0.6 on 3500 cores while Charm++ is slightly below. Figure 18b visualizes the relative fraction of the most relevant algorithm phases in a single run. It clearly identifies histogramming as the bottleneck if we scale up the number of processors. This is not surprising because with 128 nodes (2048 ranks) each rank operates on only 8 MB of memory.

[4] v6.9.0, http://charmplusplus.org/download/.

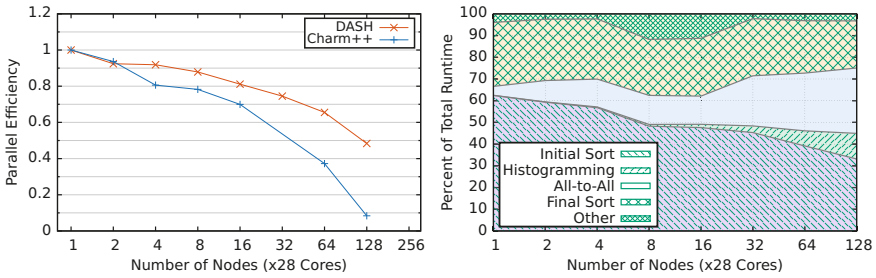


Fig. 19 Weak scaling study with Charm++ and DASH. (**a**) Weak scaling efficiency. (**b**) Weak scaling behavior of `dash::sort`

Figure 19a depicts the weak scaling efficiency. The absolute median execution time for DASH started from 2.3 s on one node and ended with 4.6 s if we scale to 128 nodes (3584 cores). As expected, the largest fraction of time is consumed in local sorting and the ALL-TO-ALL data exchange because we have to communicate 256 GB across the network. Figure 19b confirms this. The collective ALLREDUCE of $P - 1$ splitters among all processors in histogramming overhead is almost amortized from the data exchange which gives an overall good scalability for DASH. The Charm++ histogramming algorithm again shows high volatility with running times from 5–25 s, resulting in drastic performance degradation.

Our implementation shows good scalability on parallel machines with a large processor count. Compared to other algorithms we do not pose any assumptions on the number of ranks, the globally allocated memory volume or the key distribution. Performance measurements reveal that our general purpose approach does not result in performance degradation compare to other state-of-the-art algorithms. Our optimized MPI ALL-TO-ALL exchange with advanced PGAS techniques shows how we can significantly improve communication-computation overlap. Finally, the STL compliant interface enables programmers to easily integrate a scalable sorting algorithm into scientific implementations.

4 Use Cases and Applications

4.1 A Productivity Study: The Cowichan Benchmarks

In this section we present an evaluation of DASH focusing on productivity and performance by comparison with four established parallel programming approaches (Go, Chapel, Cilk, TBB) using the Cowichan set of benchmark kernels.

www.dbooks.org

4.1.1 The Cowichan Problems

The Cowichan problems [36], named after a tribal area in the Canadian Northwest, is a set of small benchmark kernels that have been developed primarily for the purpose of assessing the usability of parallel programming systems. There are two versions of the Cowichan problems and here we restrict ourselves to a subset of the problems found in the second set. The comparison presented in this section is based on previous work by Nanz et al. [23] as we use their publicly available code[5] to compare with our own implementation of the Cowichan benchmarks using DASH. The code developed as part of a study by Nanz et al. has been created by expert programmers in Go, Chapel, Cilk and TBB and can thus be regarded as idiomatic for each approach and free of obvious performance defects.

The five (plus one) problems we consider in our comparison are the following:

randmat: Generate a (`nrows` × `ncols`) matrix `mat` of random integers in the range $0, \ldots,$ `max` $- 1$ using a deterministic pseudo-random number generator (PRNG).

thresh: Given an integer matrix `mat`, and a thresholding percentage `p`, compute a boolean matrix `mask` of similar size, such that `mask` selects `p` percent of the largest values of `mat`.

winnow: Given an integer matrix `mat`, a boolean matrix `mask`, and a desired number of target elements `nelem`, perform a *weighted point selection* operation using sorting and selection.

outer: Given a vector of `nelem` (row, col) points, compute an (`nelem` × `nelem`) *outer product* matrix `omat` and a vector `vec` of floating point values based on the Euclidean distance between the points and the origin, respectively.

matvec: Given an `nelem` × `nelem` matrix `mat` and a vector `vec`, compute the matrix-vector product (row-by-row inner product) `res`.

chain: Combine the kernels in a sequence such that the output of one becomes the input for the next. I.e., chain = randmat ∘ thresh ∘ winnow ∘ outer ∘ matvec.

4.1.2 The Parallel Programming Approaches Compared

We compare our implementation of the Cowichan problems with existing solutions in the following four programming approaches.

Chapel [8] is an object-oriented partitioned global address space (PGAS) programming language developed since the early 2000s by Cray, originally as part of DARPA's High Productivity Computing Systems (HPCS) program. We have used Chapel version 1.15.0 in our experiments.

Go [10] is a general-purpose systems-level programming language developed at Google in the late 2000s that focuses on concurrency as a first-class concern. Go supports lightweight threads called *goroutines* which are invoked by prefixing

[5] https://bitbucket.org/nanzs/multicore-languages/src.

a function call with the `go` keyword. Channels provide the idiomatic way for communication between goroutines but since all goroutines share a single address space, pointers can also be used for data sharing. We have used Go version 1.8 in our experiments.

Cilk [6] started as an academic project at MIT in the 1990s. Since the late 2000s the technology has been extended and integrated as Cilk Plus into the commercial compiler offerings from Intel and more recently open source implementations for the GNU Compiler Collection (GCC) and LLVM became available. Cilk's initial focus was on lightweight tasks invoked using the `spawn` keyword and dynamic workstealing. Later a parallel loop construct (`cilk_for`) was added. We have used Cilk as integrated in Intel C/C++ compilers version 18.0.2.

Intel Threading Building Blocks (TBB) [25] is a C++ template library for parallel programming that provides tasks, parallel algorithms and containers using a work-stealing approach that was inspired by the early work on Cilk. We have used TBB version 2018.0 in our experiments, which is part of Intel Parallel Studio XE 2018.

4.1.3 Implementation Challenges and DASH Features Used

In this section we briefly describe the challenges encountered when implementing the Cowichan problems, a more detailed discussion can be found in a recent publication [13]. Naturally this small set of benchmarks only exercises a limited set of the features offered by either programming approach. However, we believe that the requirements embedded in the Cowichan problems are relevant to a wide set of other uses cases, including the classic HPC application areas.

Memory Allocation and Data Structure Instantiation The Cowichan problems use one- and two-dimensional arrays as the main data structures. 1D arrays are widely supported by all programming systems. True multidimensional arrays, however, are not universally available and as a consequence workarounds are commonly used. The Cilk and TBB implementation both adopt a linearized representation of the 2D matrix and use a single malloc call to allocate the whole matrix. Element-wise access is performed by explicitly computing the offset of the element in the linearized representation by $mat[i * ncols + j]$. Go uses a similar approach but bundles the dimensions together with the allocated memory in a custom type. In contrast, Chapel and DASH support a concise and elegant syntax for the allocation and direct element-wise access of their built-in multidimensional arrays. In the case of DASH, the distributed multidimensional array is realized as a C++ template class that follows the container concept of the standard template library (STL) [11].

Work Sharing In all benchmarks, work has to be distributed between multiple processes or threads, for example when computing the random values in *randmat* in parallel. *randmat* requires that the result be independent of the degree of parallelism used and all implementations solve this issue by using a separate deterministic seed value for each row of the matrix. A whole matrix row is the unit of work that is

www.dbooks.org

distributed among the processes or threads. The same strategy is also used for *outer* and *product*.

Cilk uses `cilk_for` to automatically partition the matrix rows and TBB uses C++ template mechanisms to achieve a similar goal. Go does not offer built-in constructs for simple work sharing and the functionality has to be laboriously created manually using goroutines, channels, and ranges.

In Chapel this type of work distribution can simply be expressed as a parallel loop (`forall`).

In DASH, the work distribution follows the data distribution. I.e., each unit is responsible for computing on the data that is locally resident, the *owner computes* model. Each unit determines its locally stored portion of the matrix (guaranteed to be a set of rows by the data distribution pattern used) and works on it independently.

Global Max Reduction In *thresh*, the largest matrix entry has to be determined to initialize other data structures to their correct size. The Go reference implementation doesn't actually perform this step and instead just uses a default size of 100, Go is thus not discussed further in this section.

In Cilk a `reducer_max` object together with a parallel loop over the rows is employed to find the maximum. Local maximal values are computed in parallel and then the global maximum is found using the reducer object. In TBB a similar construct is used (`tbb::parallel_reduce`). In these approaches finding the local maximum and computing the global maximum are separate steps that require a considerable amount of code (several 10s of lines of code).

Chapel again has the most concise syntax of all approaches, the maximum value is found simply by `nmax = max reduce matrix`. The code in the DASH solution is nearly as compact, by using the `max_element()` algorithm to find the maximum. Instead of specifying the matrix object directly, in DASH we have to couple the algorithm and the container using the iterator interface by passing `mat.begin()` and `mat.end()` to denote the range of elements to be processed.

Parallel Histogramming *thresh* requires the computation of a global cumulative histogram over an integer matrix. Thus, for each integer value $0, \ldots,$ `nmax` $- 1$ we need to determine the number of occurrences in the given matrix in parallel. The strategy used by all implementations is to compute one or multiple histograms by each thread in parallel and to later combine them into a single global histogram.

In DASH we use a distributed array to compute the histogram. First, each unit computes the histogram for the locally stored data, by simply iterating over all local matrix elements and updating the local histogram (`histo.local`). Then `dash::transform` is used to combine the local histograms into a single global histogram located at unit 0. `dash::transform` is modeled after `std::transform`, a mutating sequence algorithm. Like the STL variant, the algorithm works with two input ranges that are combined using the specified operation into an output range.

Parallel Sorting *winnow* requires the sorting of 3-tuples using a custom comparison operator. Cilk and TBB use a parallel shared memory sort. Go and Chapel

call their respective internal sort implementations (quicksort in the case of Chapel) which appears to be unparallelized. DASH can take advantage of a parallel and distributed sort implementation based on histogram sort cf. 3.2.

4.1.4 Evaluation

We first compare the productivity of DASH compared to the other parallel programming approaches before analyzing the performance differences.

Productivity We evaluate programmer productivity by analyzing source code complexity. Table 1 shows the lines of code (LOC) used in the implementation for each kernel, counting only lines that are not empty or comments. Of course, LOC is a crude approximation for source code complexity but few other metrics are universally accepted or available for different programming languages. LOC at least gives a rough idea for source code size, and, as a proxy, development time, likelihood for programming errors and productivity. The overall winners in the productivity category are Chapel and Cilk, which achieve the smallest source code size for three benchmark kernels. For most kernels, DASH also achieves a remarkably small source code size considering that the same source code can run on shared memory as well as on distributed memory machines.

Performance As the hardware platform for our experiments we have used one or more nodes of SuperMUC Phase 2 (SuperMUC-HW) with Intel Xeon E5-2697 (Haswell) CPUs with 2.6 GHz, 28 cores and 64 GB of main memory per node.

Single Node Comparison We first investigate the performance differences between DASH and the four established parallel programming models on a single Haswell node. We select a problem size of nrows = ncols = 30,000 because this is the largest size that successfully ran with all programming approaches.

Table 2 shows the absolute performance (runtime in seconds) and relative runtime (compared to DASH) when using all 28 cores of a single node. Evidently, DASH is the fastest implementation, followed by Cilk and TBB. Chapel and especially Go can not deliver competitive performance in this setting.

Analyzing the scaling behavior in more detail (not shown graphically due to space restrictions) by increasing the number of cores used on the system from 1 to 28 reveals a few interesting trends. For outer and randmat, DASH, Cilk and TBB

Table 1 Lines-of-code (LOC) measure for each kernel and programming approach, counting non-empty and non-comment lines only

	DASH	Go	Chapel	TBB	Cilk
Randmat	18	29	14	15	12
Thresh	31	63	30	56	52
Winnow	67	94	31	74	78
Outer	23	38	15	19	15
Product	19	27	11	14	10

www.dbooks.org

Table 2 Performance comparison for each kernel and programming approach using all cores on one node of SuperMUC-HW

	Absolute runtime (sec.)					Relative runtime				
	DASH	Go	Chapel	TBB	Cilk	DASH	Go	Chapel	TBB	Cilk
Randmat	0.12	5.84	0.36	0.15	0.19	1.00	48.69	3.03	1.26	1.58
Thresh	0.20	0.64	0.53	0.41	0.40	1.00	3.19	2.64	2.06	2.00
Winnow	2.99	366.13	256.40	9.45	4.28	1.00	122.45	85.75	3.16	1.43
Outer	0.25	1.18	0.39	0.27	0.31	1.00	4.70	1.56	1.06	1.24
Product	0.06	0.46	0.19	0.12	0.13	1.00	7.66	3.15	2.01	2.16

behave nearly identical in terms of absolute performance and scaling behavior. The small performance advantage of DASH can be attributed to better NUMA locality of DASH, where all work is done on process-local data. For thresh and winnow DASH can take advantage of optimized parallel algorithms (for global max reduction and sorting, respectively) whereas these operations are performed sequentially in some of the other approaches. For product the DASH implementation takes advantage of a local copy optimization to improve data locality. The scaling study reveals that this optimization at first costs performance but pays off at larger core counts.

Multinode Scaling We next investigate the scaling of the DASH implementation on up to 16 nodes (448 total cores) of SuperMUC-HW. None of the other approaches can be compared with DASH in this scenario. Cilk and TBB are naturally restricted to shared memory systems by their threading-based nature. Go realizes the CSP (communicating sequential processes) model that would, in principle, allow for a distributed memory implementation but since data sharing via pointers is allowed, Go is also restricted to a single shared memory node. Finally, Chapel targets both shared and distributed memory systems, but the implementation of the Cowichan problems available in this study is not prepared to be used with multiple locales and cannot make use of multiple nodes (it lacks the `dmapped` specification for data distribution).

The scaling results are shown in Fig. 20 for two data set sizes. In Fig. 20 (left) we show the speedup relative to one node for a small problem size (nrows = ncols = 30,000) and in Fig. 20 (right) we show the speedup of a larger problem size (nrows = ncols = 80,000) relative to two nodes, since this problem is too big to fit into the memory of a single node.

Evidently for the smaller problem size, the benchmark implementations reach their scaling limit at about 10 nodes, whereas the larger problem sizes manage to scale well even to 16 nodes, with the exception of the product benchmark which shows the worst scaling behavior. This behavior can be explained by the relatively large communication requirement of the product benchmark kernel.

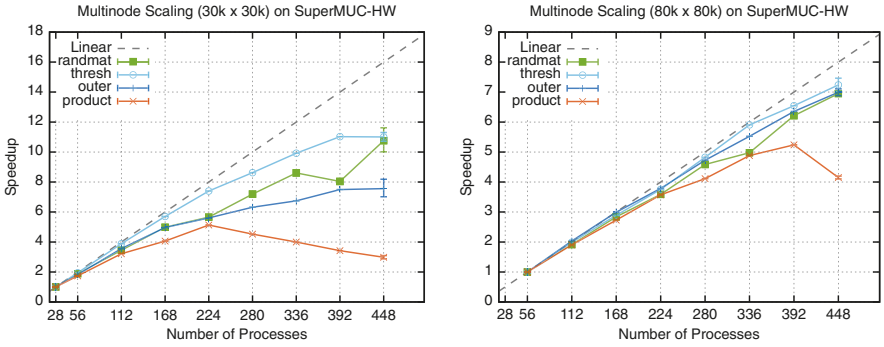


Fig. 20 Scaling behavior of the Cowichan benchmarks with up to 16 nodes on SuperMUC-HW. (**a**) Multinode Scaling, 30×30 k Matrix. (**b**) Multinode Scaling: $80k \times 80k$ Matrix

4.1.5 Summary

In this section we have evaluated DASH, a new realization of the PGAS approach in the form of a C++ template library by comparing our implementation of the Cowichan problems with those developed by expert programmers in Cilk, TBB, Chapel, and Go. We were able to show that DASH achieves both remarkable performance and productivity that is comparable with established shared memory programming approaches. DASH is also the only approach in our study where the same source code can be used both on shared memory systems and on scalable distributed memory systems. This step, from shared memory to distributed memory systems is often the most difficult for parallel programmers because it frequently goes hand in hand with a re-structuring of the entire data distribution layout of the application. With DASH the same application can seamlessly scale from a single shared memory node to multiple interconnecting nodes.

4.2 Task-Based Application Study: LULESH

The Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is part of the Department of Energy's Coral proxy application benchmark suite [19]. The domain typically scales with the number of processes and is divided into a grid of nodes, elements, and regions of elements. The distribution and data exchange follows a 27-point stencil with three communication steps per timestep.

The reference implementation of LULESH uses MPI send/recv communication to facilitate the boundary exchange between neighboring processes and OpenMP worksharing constructs are used for shared memory parallelization. Instead, we

www.dbooks.org

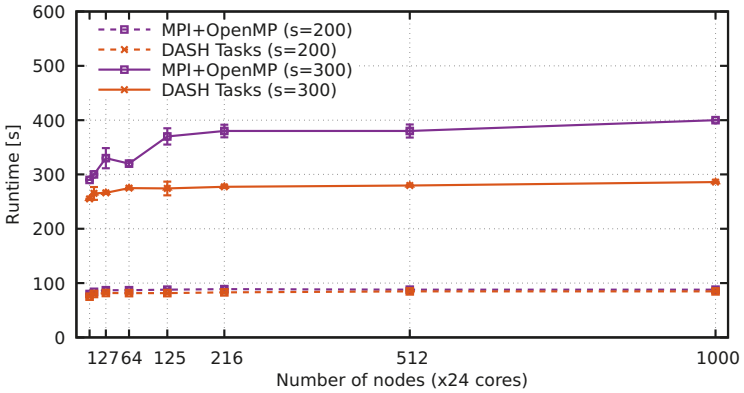


Fig. 21 Lulesh reference implementation and Lulesh using DASH tasks running on a Cray XC40

iteratively ported LULESH to use DASH distributed data structures for neighbor communication and DASH tasks for shared memory parallelization, with the ability to partially overlap communication and computation.

The first step was to adapt the communication to use DASH distributed data structures instead of MPI [12]. In order to gradually introduce tasks, we started by porting the OpenMP worksharing loop constructs with to use the DASH task-loop construct discussed in Sect. 2.1.3. In a second step, the iteration chunks of the taskloops were connected through dependencies, allowing a breadth-first scheduler to execute tasks from different task-loop statements concurrently. In a last step, a set of high-level tasks has been introduced that encapsulate the task-loop statements and coordinate the computation and communication tasks.

The resulting performance at scale on a Cray XC40 is shown in Fig. 21. For larger problem sizes ($s = 300^3$ elements per node), the speedup at scale of the DASH port over the reference implementation is about 25%. For smaller problem sizes ($s = 200^3$), the speedup is significantly smaller at about 5%. We believe that further optimizations in the tasking scheduler may yield improvements even for smaller problem sizes.

5 Outlook and Conclusion

We have presented an overview of our parallel programming approach DASH, focusing on recent activities in the areas of support for task-based execution, dynamic locality, parallel algorithms, and smart data structures. Our results show that DASH offers a productive programming environment that also allows programmers to write highly efficient and scalable programs with performance on-par or exceeding solutions relying on established programming systems.

Work on DASH is not finished. The hardware landscape in high performance computing is getting still more complex, while the application areas are getting more diverse. Heterogeneous compute resources are common, nonvolatile memories are making their appearance and domain specific architectures are on the horizon. These and other challenges must be addressed by DASH to be a viable parallel programming approach for many users.

The challenge of utilizing heterogeneous computing resources, primarily in the form of graphics processing units (GPUs) used in high performance computing systems, is addressed in a project building on DASH funded by the German Federal Ministry of Education and Research (BMBF) called MEPHISTO. We close our report on DASH with a short discussion of MEPHISTO.

5.1 MEPHISTO

The PGAS model simplifies the implementation of programs for distributed memory plattforms, but data locality plays an important role for performance critical applications. The *owner-computes* paradigm aims to maximize the performance by minimizing the intra-node data movement. This focus on locality also encourages the usage of shared memory parallelism. During the DASH project the partners already experimented with shared memory parallelism and how it can be integrated into the DASH ecosystem. OpenMP was integrated into suitable algorithms, experiments with Intel's Thread Building Blocks as well as conventional POSIX-thread-based parallelism were conducted. These, however, had to be fixed for a given algorithm: a user had no control over the acceleration and the library authors had to find sensible configurations (e.g. level of parallelism, striding, scheduling). Giving users of the DASH library more possibilities and flexibility is a focus of the MEPHISTO project.

The MEPHISTO project partly builds on the work that has been done in DASH. One of its goal is to integrate abstractions for better data locality and heterogeneous programming with DASH. Within the scope of MEPHISTO two further projects are being integrated with DASH:

- Abstraction Library for Parallel Kernel Acceleration (ALPAKA) [22]
- Low Level Abstraction of Memory Access[6] (LLAMA)

ALPAKA is a library that adds abstractions for node-level parallelism for C++ programs. Once a kernel is written with ALPAKA, it can be easily ported to different accelerators and setups. For example a kernel written with ALPAKA can be executed in a multi-threaded CPU environment as well as on a GPU, for example using the CUDA backend. ALPAKA provides optimized code for each accelerator that can be further customized by developers. To switch from one back end to

[6] https://github.com/mephisto-hpc/llama.

www.dbooks.org

another requires just the change of one type definition in the code so that the code is portable.

Integrating ALPAKA and DASH brings flexibility for node-level parallelism within a PGAS environment: switching an algorithm from a multi threaded CPU implementation to an accelerator now only requires passing one more parameter to the function call. It also gives the developer an interface to control *how* and *where* the computation should happen. The kernels of algorithms like `dash::transform_reduce` are currently extended to work with external *executors* like ALPAKA. Listing 5 shows how executors are used to offload computation using ALPAKA. Note that the interface allows offloading to other implementations as well. In the future DASH will support any accelerator that implements a simple standard interface for node-level parallelism.

```
policy.executor().bulk_twoway_execute(
  [=](size_t      block_index,
      size_t      element_index,
      T*          res,
      value_type* block_first) {
    res[block_index] = binary_op(
                         res[block_index],
                         unary_op(block_first[element_index])
    ↪ );
  },
  in_first, // a "shape"
  [&]() -> std::vector<std::future<T>>& {
    return results;
  },
  std::ignore); // shared state (unused)
```

Listing 5 Offloading using an executor inside `dash::transform_reduce`. The executor can be provided by MEPHISTO

LLAMA on the other hand focuses solely on the format of data in memory. DASH already provides a flexible *Pattern Concept* to define the data placement for a distributed container. However, LLAMA gives developers finer grained control over the data layout. DASH's patterns map elements of a container to locations in memory, but the layout of the elements itself is fixed. With LLAMA, developers can specify the data layout with a C++ Domain Specific Language (DSL) to fit the application's needs. A typical example is a conversion from Structure of Arrays (SoA) to Array of Structures (AoS) and vice versa. But also more complex transformations like projections are being evaluated.

Additionally to the integration of LLAMA, a more flexible, hierarchical and context-sensitive pattern concept is being evaluated. Since the current patterns map elements to memory locations in terms of units (i.e., MPI processes), using other sources for parallelism can be a complex task. Mapping elements to (possibly multiple) accelerators was not easily possible. Local patterns extend the existing patterns. By matching elements with *entities* (e.g. a GPU), the node-local data may be assigned to other compute units beside processes.

Both projects are currently evaluated in the context of DASH to explore how the PGAS programming model can be used more flexibly and efficiently.

Acknowledgments The authors would like to thank the many dedicated and talented individuals who contributed to the success of the DASH project. DASH started out as "just an idea" that was turned into reality with the help of Benedikt Bleimhofer, Daniel Rubio Bonilla, Daniel Diefenthaler, Stefan Effenberger, Colin Glass, Kamran Idrees, Benedikt Lehmann, Michael Maier, Matthias Maiterth, Yousri Mhedheb, Felix Mössbauer, Sebastian Oeste, Bernhard Saumweber, Jie Tao, Bert Wesarg, Huan Zhou, Lei Zhou.

We would also like to thank the German research foundation (DFG) for the funding received through the SPPEXA priority programme and initiators and managers of SPPEXA for their foresight and level-headed approach.

References

1. Agullo, E., Aumage, O., Faverge, M., Furmento, N., Pruvost, F., Sergent, M., Thibault, S.P.: Achieving high performance on supercomputers with a sequential task-based programming model. IEEE Trans. Parallel Distrib. Syst. (2018). https://doi.org/10.1109/TPDS.2017.2766064
2. Axtmann, M., Bingmann, T., Sanders, P., Schulz, C.: Practical massively parallel sorting. In: Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures, pp. 13–23. ACM, New York (2015)
3. Ayguade, E., Copty, N., Duran, A., Hoeflinger, J., Lin, Y., Massaioli, F., Teruel, X., Unnikrishnan, P., Zhang, G.: The design of OpenMP tasks. IEEE Trans. Parallel Distrib. Syst. **20**, 404–418 (2009). https://doi.org/10.1109/TPDS.2008.105
4. Bauer, M., Treichler, S., Slaughter, E., Aiken, A.: Legion: expressing locality and independence with logical regions. In: 2012 International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pp. 1–11 (2012). https://doi.org/10.1109/SC.2012.71
5. Blelloch, G.E., Leiserson, C.E., Maggs, B.M., Plaxton, C.G., Smith, S.J., Zagha, M.: A comparison of sorting algorithms for the connection machine cm-2. In: Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, pp. 3–16. ACM, New York (1991)
6. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: An Efficient Multithreaded Runtime System, vol. 30. ACM, New York (1995)
7. Bosilca, G., Bouteiller, A., Danalis, A., Herault, T., Lemariner, P., Dongarra, J.: Dague: a generic distributed dag engine for high performance computing. In: Proceedings of the Workshops of the 25th IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2011 Workshops), pp. 1151–1158 (2011)
8. Chamberlain, B.L., Callahan, D., Zima, H.P.: Parallel programmability and the Chapel language. Int. J. High Perform. Comput. Appl. **21**, 291–312 (2007)
9. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., Von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. ACM SIGPLAN Not. **40**, 519–538 (2005)
10. Donovan, A.A., Kernighan, B.W.: The Go Programming Language, 1st edn. Addison-Wesley Professional, Boston (2015)
11. Fuchs, T., Fürlinger, K.: A multi-dimensional distributed array abstraction for PGAS. In: Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016), Sydney, pp. 1061–1068 (2016). https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0150

www.dbooks.org

12. Fürlinger, K., Fuchs, T., Kowalewski, R.: DASH: a C++ PGAS library for distributed data structures and parallel algorithms. In: Proceedings of the 18th IEEE International Conference on High Performance Computing and Communications (HPCC 2016), Sydney, pp. 983–990 (2016). https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0140
13. Fürlinger, K., Kowalewski, R., Fuchs, T., Lehmann, B.: Investigating the performance and productivity of DASH using the cowichan problems. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region Workshops, Tokyo (2018). https://doi.org/10.1145/3176364.3176366
14. Grossman, M., Kumar, V., Budimlić, Z., Sarkar, V.: Integrating asynchronous task parallelism with OpenSHMEM. In: Workshop on OpenSHMEM and Related Technologies, pp. 3–17. Springer, Berlin (2016)
15. Harsh, V., Kalé, L.V., Solomonik, E.: Histogram sort with sampling. CoRR abs/1803.01237 (2018). http://arxiv.org/abs/1803.01237
16. Helman, D.R., JáJá, J., Bader, D.A.: A new deterministic parallel sorting algorithm with an experimental evaluation. ACM J. Exp. Algorithmics **3**, 4 (1998)
17. Kaiser, H., Brodowicz, M., Sterling, T.: Parallex an advanced parallel execution model for scaling-impaired applications. In: 2009 International Conference on Parallel Processing Workshops (2009). https://doi.org/10.1109/ICPPW.2009.14
18. Kaiser, H., Heller, T., Adelstein-Lelbach, B., Serio, A., Fey, D.: HPX: a task based programming model in a global address space. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS '14. ACM, New York (2014). http://doi.acm.org/10.1145/2676870.2676883
19. Karlin, I., Keasler, J., Neely, R.: Lulesh 2.0 updates and changes. Technical Report. LLNL-TR-641973 (2013)
20. Kowalewski, R., Jungblut, P., Fürlinger, K.: Engineering a distributed histogram sort. In: 2019 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, Piscataway (2019)
21. Kumar, V., Zheng, Y., Cavé, V., Budimlić, Z., Sarkar, V.: HabaneroUPC++: a compiler-free PGAS library. In: Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS '14. ACM, New York (2014). https://doi.org/10.1145/2676870.2676879
22. Matthes, A., Widera, R., Zenker, E., Worpitz, B., Huebl, A., Bussmann, M.: Tuning and optimization for a variety of many-core architectures without changing a single line of implementation code using the alpaka library (2017). https://arxiv.org/abs/1706.10086
23. Nanz, S., West, S., Da Silveira, K.S., Meyer, B.: Benchmarking usability and performance of multicore languages. In: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 183–192. IEEE, Piscataway (2013)
24. OpenMP Architecture Review Board: OpenMP Application Programming Interface, Version 5.0 (2018). https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf
25. Reinders, J.: Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism. O'Reilly & Associates, Sebastopol (2007)
26. Robison, A.D.: Composable parallel patterns with intel cilk plus. Comput. Sci. Eng. **15**, 66–71 (2013). https://doi.org/10.1109/MCSE.2013.21
27. Saraswat, V., Almasi, G., Bikshandi, G., Cascaval, C., Cunningham, D., Grove, D., Kodali, S., Peshansky, I., Tardieu, O.: The asynchronous partitioned global address space model. In: The First Workshop on Advances in Message Passing, pp. 1–8 (2010)
28. Saukas, E., Song, S.: A note on parallel selection on coarse-grained multicomputers. Algorithmica **24**(3-4), 371–380 (1999)
29. Shirako, J., Peixotto, D.M., Sarkar, V., Scherer, W.N.: Phasers: a unified deadlock-free construct for collective and point-to-point synchronization. In: Proceedings of the 22nd Annual International Conference on Supercomputing, ICS '08. ACM, New York (2008). https://doi.org/10.1145/1375527.1375568
30. Slaughter, E., Lee, W., Treichler, S., Bauer, M., Aiken, A.: Regent: a high-productivity programming language for HPC with logical regions. In: SC15: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–12 (2015). https://doi.org/10.1145/2807591.2807629

31. Sundar, H., Malhotra, D., Biros, G.: Hyksort: a new variant of hypercube quicksort on distributed memory architectures. In: Proceedings of the 27th international ACM conference on International Conference on Supercomputing, pp. 293–302. ACM, New York (2013)
32. Tejedor, E., Farreras, M., Grove, D., Badia, R.M., Almasi, G., Labarta, J.: A high-productivity task-based programming model for clusters. Concurr. Comp. Pract. Exp. **24**, 2421–2448 (2012). https://doi.org/10.1002/cpe.2831
33. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of collective communication operations in MPICH. Int. J. High Perform. Comput. Appl. **19**(1), 49–66 (2005). https://doi.org/10.1177/1094342005051521
34. Tillenius, M.: SuperGlue: a shared memory framework using data versioning for dependency-aware task-based parallelization. SIAM J. Sci. Comput. **37**, C617–C642 (2015). http://epubs.siam.org/doi/10.1137/140989716
35. Tsugane, K., Lee, J., Murai, H., Sato, M.: Multi-tasking execution in pgas language xcalablemp and communication optimization on many-core clusters. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region. ACM, New York (2018). https://doi.org/10.1145/3149457.3154482
36. Wilson, G.V., Irvin, R.B.: Assessing and comparing the usability of parallel programming systems. University of Toronto. Computer Systems Research Institute (1995)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

ESSEX: Equipping Sparse Solvers For Exascale



Christie L. Alappat, Andreas Alvermann, Achim Basermann, Holger Fehske, Yasunori Futamura, Martin Galgon, Georg Hager, Sarah Huber, Akira Imakura, Masatoshi Kawai, Moritz Kreutzer, Bruno Lang, Kengo Nakajima, Melven Röhrig-Zöllner, Tetsuya Sakurai, Faisal Shahzad, Jonas Thies, and Gerhard Wellein

Abstract The ESSEX project has investigated programming concepts, data structures, and numerical algorithms for scalable, efficient, and robust sparse eigenvalue solvers on future heterogeneous exascale systems. Starting without the burden of legacy code, a holistic performance engineering process could be deployed across the traditional software layers to identify efficient implementations and guide sustainable software development. At the basic building blocks level, a flexible MPI+X programming approach was implemented together with a new sparse data structure (SELL-C-σ) to support heterogeneous architectures by design. Furthermore, ESSEX focused on hardware-efficient kernels for all relevant architectures and efficient data structures for block vector formulations of the eigensolvers. The algorithm layer addressed standard, generalized, and nonlinear eigenvalue problems and provided some widely usable solver implementations including a block Jacobi–Davidson algorithm, contour-based integration schemes, and filter polynomial approaches. Adding to the highly efficient kernel implementations, algorithmic advances such as adaptive precision, optimized filtering coefficients, and preconditioning have further improved time to solution. These developments

C. L. Alappat · G. Hager · M. Kreutzer · F. Shahzad · G. Wellein (✉)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany

M. Röhrig-Zöllner · J. Thies · A. Basermann
German Aerospace Center, Cologne, Germany

M. Galgon · S. Huber · B. Lang
Bergische Universität Wuppertal, Wuppertal, Germany

A. Alvermann · H. Fehske
Universität Greifswald, Greifswald, Germany

M. Kawai · K. Nakajima
University of Tokyo, Tokyo, Japan

Y. Futamura · A. Imakura · T. Sakurai
University of Tsukuba, Tsukuba, Japan

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_7

were guided by quantum physics applications, especially from the field of topological insulator- or graphene-based systems. For these, ScaMaC, a scalable matrix generation framework for a broad set of quantum physics problems, was developed. As the central software core of ESSEX, the PHIST library for sparse systems of linear equations and eigenvalue problems has been established. It abstracts algorithmic developments from low-level optimization. Finally, central ESSEX software components and solvers have demonstrated scalability and hardware efficiency on up to 256 K cores using million-way process/thread-level parallelism.

1 Introduction

The efficient solution of linear systems or eigenvalue problems involving large sparse matrices has been an active research field in parallel and high performance computing for many decades. Software packages like Trilinos [33] or PETSc [9] have been developed to great maturity, and algorithmic improvements were accompanied by advances in programming abstractions addressing, e.g., node-level heterogeneity (cf. Kokkos [19]). Completely new developments such as Ginkgo[1] are rare and do not focus on large-scale applications or node-level efficiency.

Despite projections from the late 2000s, hardware architectures have not developed away from traditional clustered multicore systems. However, a clear trend of increased node-level parallelism and heterogeneity has been observed. Although several new architectures entered the field (and some vanished again), the basic concepts of core-level code execution and data parallelism have not changed. This is why the MPI+X concept is still a viable response to the challenge of hardware diversity.

Performance analysis of highly parallel code typically concentrated on scalability, but provably optimal node-level performance was rarely an issue. Moreover, strong abstraction boundaries between linear algebra building blocks, solvers, and applications made it hard to get a holistic view on a minimization of time to solution, encompassing optimizations in the algorithmic and implementation dimensions.

In this setting, the ESSEX project took the opportunity to start from a clean slate, deliberately breaking said abstraction boundaries to investigate performance bottlenecks together with algorithmic improvements from the core to the highly parallel level. Driven by the targeted application fields, bespoke solutions were developed for selected algorithms and applications. The experience gained in the development process will lead the way towards more generic approaches rather than compete with established libraries in terms of generality. The overarching motif was a consistent performance engineering process that coordinated all performance-relevant activities across the different software layers [1, 3, 4, 20, 52–54, 56, 62, 64].

[1] https://github.com/ginkgo-project/ginkgo.

www.dbooks.org

Consequently, the ESSEX parallel building blocks layer implemented in the GHOST library [55] supports MPI+X, with X being a combination of node-level programming models able to fully exploit hardware heterogeneity, functional parallelism, and data parallelism. Despite fluctuations in hardware architectures and new programming models hitting the market every year, OpenMP or CUDA is still the most promising and probably most sustainable choice for X, and ESSEX-II adhered to it. In addition, engineering highly specialized kernels including sparse-matrix multiple-vector operations and appropriate data structures for all relevant compute architectures provided the foundation for hardware- and energy-efficient large-scale computations.

Building on these high-performance building blocks, one focus of the algorithm layer was put on the block formulation of Jacobi–Davidson [64] and filter diagonalization [56] methods, the hardware efficiency of preconditioners [46–48], and the development of hardware-aware coloring schemes [1]. In terms of scalability, the project has investigated new contour-based integration eigensolvers [23, 24] that can exploit additional parallelism layers beyond the usual data parallelism. The solvers developed in ESSEX can tackle standard, generalized, and nonlinear eigenvalue problems and may also be used to extract large bulks of extremal and inner eigenvalues.

The applications layer applies the algorithms and building blocks to deliver scalable solutions for topical quantum materials like graphene, topological insulators, or superconductors, and nonlinear dynamical systems like reaction-diffusion systems. A key issue for large-scale simulations is the scalable (in terms of size and parallelism) generation of the sparse matrix representing the model Hamiltonian. Our matrix generation framework ScaMaC can be integrated into application code to allow the on-the-fly, in-place construction of the sparse matrix. Beyond the ESSEX application fields, matrices from many other relevant areas can be produced by ScaMaC.

The PHIST library [78] is the sustainable outcome of the performance-centric efforts in ESSEX. It is built on a rigorous software and performance engineering process, comprises diverse solver components, and supports multiple backends (e.g., Trilinos, PETSc, ESSEX kernels). It also interfaces to multiple languages such as C, C++, Fortran 2003, and Python. The CRAFT library [73] provides user-friendly access to fault tolerance via checkpoint/restart and automatic recovery for iterative codes using standard C++.

Scalabilty, performance, and portability have been tested on three top-10 supercomputers covering the full range of architecures available during the ESSEX project time frame: Piz Daint[2] (heterogeneous CPU-GPU), OakForest-PACS[3] (many-core), and SuperMUC-NG[4] (standard multi-core).

[2] https://www.cscs.ch/computers/piz-daint/.

[3] https://www.cc.u-tokyo.ac.jp/en/supercomputer/ofp/service/.

[4] https://doku.lrz.de/display/PUBLIC/SuperMUC-NG.

This review focuses on important developments in ESSEX-II. After presenting a brief overview of the most relevant achievements in the first project phase ESSEX-I in Sects. 2 and 3 details algorithmic developments in ESSEX-II, notably with respect to preconditioners and projection-based methods for obtaining inner eigenvalues. Moreover, we present the RACE (Recursive Algebraic Coloring Engine) method, which delivers hardware-efficient graph colorings for parallelization of algorithms and kernels with data dependencies. In Sect. 4 we showcase performance and parallel efficiency numbers for library components developed in ESSEX-II that are of paramount importance for the application work packages: GPGPU-based tall and skinny matrix-matrix multiplication and the computation of inner eigenvalues using polynomial filter techniques. Section 5 describes the software packages that were developed to a usable and sustainable state, together with their areas of applicability. In Sect. 6 we show application results from the important areas of quantum physics and nonlinear dynamical systems. Finally, in Sect. 7 we highlight the collaborations sparked and supported by SPPEXA through the ESSEX-II project.

2 Summary of the ESSEX-I Software Structure

The Exascale-enabled Sparse Solver Repository (ESSR) was developed along the requirements of the algorithms and applications under investigation in ESSEX. It was not intended as a full-fledged replacement of existing libraries like Trilinos[5] [33], but rather as a toolbox that can supply developers with blueprints as a starting point for their own developments. In ESSEX-I, the foundations for a sustainable software framework were laid. See Sect. 5 for developments in ESSEX-II.

The initial version of the ESSR [77] comprised four components:

- GHOST (General, Hybrid and Optimized Sparse Toolkit) [55], a library of basic sparse and dense linear algebra building blocks that are not available in this form in other software packages. The development of GHOST was strictly guided by performance engineering techniques; implementations of standard kernels such as sparse matrix-vector multiplication (spMVM) and sparse matrix-multiple-vector multiplication (spMMVM) as well as tailor-made fused kernels, for instance those employed in the Kernel Polynomial Method (KPM) [81], were modeled using the roofline model. GHOST supports, by design, strongly heterogeneous environments using the MPI+X approach. See [51] for a comprehensive overview of GHOST and its building blocks.
- ESSEX-Physics, a collection of prototype implementations of polynomial eigensolvers such as the KPM and Chebyshev Filter Diagonalization (ChebFD). These were implemented on top of GHOST using tailored kernels and were shown to perform well on heterogeneous CPU-GPU systems [53].

[5] https://trilinos.org/.

www.dbooks.org

- PHIST (Pipelined Hybrid-parallel Iterative Solver Toolkit), which comprises Jacobi–Davidson type eigensolvers and Krylov methods for linear systems. One important component is a test framework that allows for continuous integration (CI) throughout the development cycle. PHIST can not only use plain GHOST as its basic linear algebra layer; it is also equipped with fallback kernel implementations and adapters for the Trilinos and Anasazi libraries. A major achievement in the development of PHIST was an efficient block Jacobi–Davidson eigenvalue solver, which could be shown to have significant performance advantages over nonblocked versions when using optimized building blocks from GHOST [64].
- BEAST (Beyond fEAST), which implements innovative projection-based eigensolvers motivated by the contour integration-based FEAST method [23]. The ESSEX-I project has contributed to improving FEAST in two ways: by proposing techniques for solving or avoiding the linear systems that arise, and by improving robustness and performance of the algorithmic scheme.

A pivotal choice for any sparse algorithm implementation is the sparse matrix storage format. In order to avoid data conversion and the need to support multiple hardware-specific formats in a single code, we developed the SELL-C-σ format [52]. It shows competitive performance on the dominating processor architectures in HPC: standard multicore server CPUs with short-vector single instruction multiple data (SIMD) capabilities, general-purpose graphics processing units (GPGPUs), and many-core designs with rather weak cores such as the Intel Xeon Phi. SELL-C-σ circumvents the performance penalties of matrices with few nonzero entries per row on architectures on which SIMD vectorization is a key element for performance even with memory-bound workloads.

In order to convert a sparse matrix to SELL-C-σ, its rows are first sorted according to their respective numbers of nonzeros. This sorting is performed across row blocks of length σ. After that, the matrix is cut into row blocks of length C. Within each block, rows are padded with zeros to equal length and then stored in column-major order. See Fig. 1 for visualizations of SELL-C-σ with $C = 6$ and $\sigma \in \{1, 12, 24\}$. Incidentally, known and popular formats can be recovered as corner cases: SELL-1-1 is the well-known CSR storage format and SELL-N-1 is ELLPACK. The particular choice of C and σ influences the performance of the spMVM operation; optimal values are typically matrix- and hardware-dependent. However, in practice one can usually find parameters that yield good performance across architectures for a particular matrix. A roofline performance model was constructed in [52] that sets an upper limit for the spMVM performance for any combination of matrix and architecture. This way, "bad" performance is easily identified. SELL-C-σ was quickly adopted by the community and is in use, in pure or adapted form, in many performance-oriented projects [5, 6, 28, 60, 80].

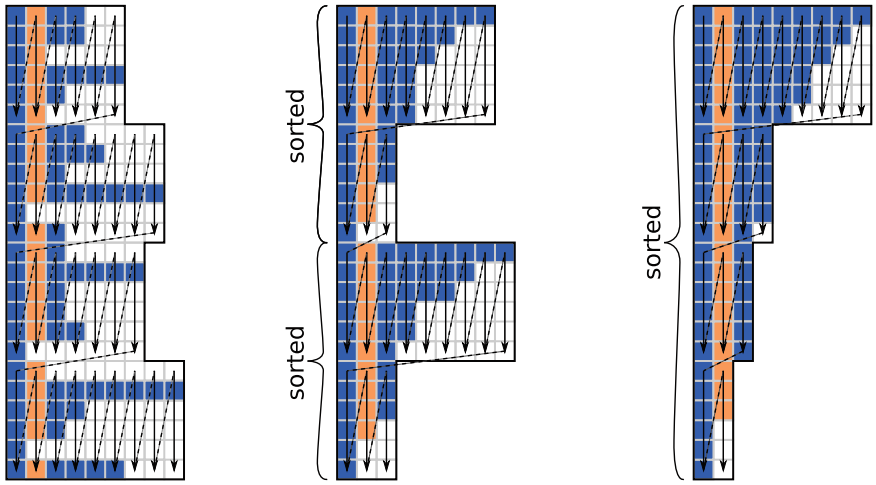


Fig. 1 Variants of the SELL-C-σ storage format. Arrows indicate the storage order of matrix values and column indices. Image from [52]. (**a**) SELL-6-1, $\beta = 0.51$. (**b**) SELL-6-12, $\beta = 0.66$. (**c**) SELL-6-24, $\beta = 0.84$

3 Algorithmic Developments

In this section we describe selected developments within ESSEX-II on the algorithmic level, in particular preconditioners for the solution of linear systems that occur in the eigensolvers, a versatile framework for computing inner eigenvalues, and a nonlinear eigensolver. We also cover a systematic comparison of contour-based methods. We close the section with the introduction of RACE, which is an algorithmic development for graph coloring guided by the constraints of hardware efficiency.

3.1 Preconditioners (ppOpen-SOL)

Two kinds of solvers have been developed: a preconditioner targeting the ill-conditioned large scale problems arising in the BEAST-C method (cf. Sect. 3.2) and a multigrid solver targeting problems arising from finite difference discretizations of partial differential equations (PDEs).

www.dbooks.org

3.1.1 Regularization

The BEAST-C method leads to a large number of ill-conditioned linear systems with complex diagonal shifts [24]. Furthermore, in many of our quantum physics applications, the system matrices have small (and sometimes random) diagonal elements. In order to apply a classic incomplete Cholesky (IC) factorization preconditioner, we used two types of regularization to achieve robustness: a blocking technique (BIC) and an additional diagonal shift [47]. Using this approach, we solved a set of 120 prototypical linear systems from this context (e.g., BEAST-C applied to quantum physics applications). Due to the complex shift, the system matrix is symmetric but not Hermitian. Hence we use an adaptation of the Conjugate Gradient (CG) method for complex symmetric matrices called COCG (conjugate orthogonal conjugate gradient [79]).

The blocking technique is a well-known approach for improving the convergence rate. In this study, we apply the technique not only for better convergence but also for more robustness. The diagonal entries in the target equations are small. By applying the blocking technique, the diagonal blocks to be inverted include larger off-diagonal entries.

The diagonal shifting is a direct measure for transforming the ill-conditioned matrices to be more diagonally dominant before performing the incomplete factorization. On the other hand, this may deteriorate the convergence of the overall method. We therefore investigate the best value for the diagonal shifting for our applications.

Figure 2 shows the effect of the regularized IC preconditioner with the COCG method. By using the diagonal shifted block IC-COCG (BIC-COCG), we solve all target linear systems.

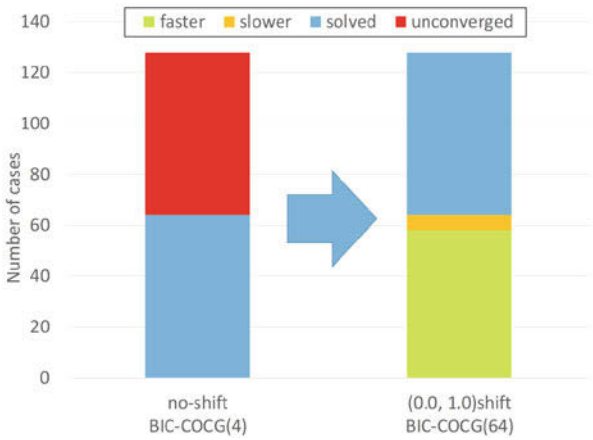


Fig. 2 Effect of the regularized IC preconditioner with the COCG method. By using the diagonal shifted block IC-COCG (BIC-COCG), we can solve all test problems from our benchmark set

3.1.2 Hierarchical Parallel Reordering

In this section, we present scalability results for the BIC preconditioner parallelized by a hierarchical parallel graph coloring algorithm. This approach yields an almost constant convergence rate with respect to the number of compute nodes, and good parallel performance.

Node-wise multi-coloring (with domain decomposition between nodes) is widely used for parallelizing IC preconditioners on clusters of shared memory CPUs. Such "localized" multi-coloring leads to a loss of robustness of the regularized IC-COCG method, and the convergence rate decreases at high levels of parallelism. To solve this problem, we parallelize the block IC preconditioner for the hybrid-parallel cluster system. In addition, we proposed the hierarchical parallelization for the multi-coloring algorithms [46]. This versatile scheme allows us to parallelize almost any multi-coloring algorithm.

Figure 3 shows the number of iterations and computational time of the BIC-COCG method on the Oakleaf-FX cluster, using up to 4,800 nodes. The benchmark matrix is the Hamiltonian of a graphene sheet simulation with more than 500 million linear equations, for which interior eigenvalues are of interest [24]. Hierarchical parallelization yields almost constant convergence with respect to the number of nodes. The computational time with 4,600 nodes is 30 times smaller than with 128 nodes, amounting to a parallel efficiency of 83.5% if the 128-node case is taken as the baseline.

3.1.3 Multiplicative Schwarz-Type Block Red-Black Gauß–Seidel Smoother

Multigrid methods are among the most useful preconditioners for elliptic PDEs. In [45] we proposed a multiplicative Schwarz block red/black Gauß–Seidel (MS-

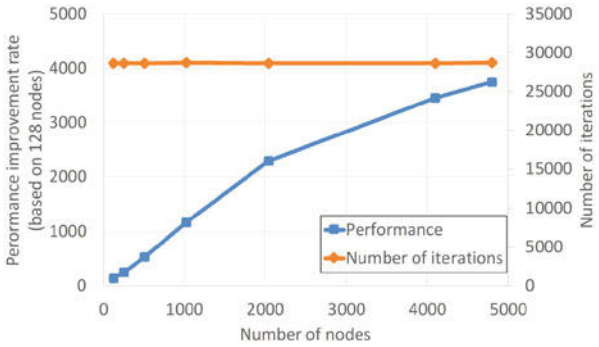


Fig. 3 Computational time and convergence of BIC-COCG for a graphene benchmark problem (strong scaling)

www.dbooks.org

		Post-smoothing									
		1	2	3	4	5	6	7	8	9	10
Pre-smoothing	1	2.74/10	2.25/8	2.30/8	2.39/8	2.21/7	2.29/7	2.38/7	2.48/7	2.59/7	2.72/7
	2	2.25/8	2.01/8	2.11/7	1.82/6	1.90/6	1.99/6	2.08/7	2.17/6	2.28/6	2.36/6
	3	2.02/7	2.08/7	1.82/6	1.87/6	1.95/6	2.03/6	2.12/6	2.17/6	2.28/6	2.36/6
	4	2.08/7	1.82/6	1.87/6	1.90/6	1.67/5	1.74/5	1.82/5	1.89/5	1.96/5	2.03/5
	5	2.15/7	1.89/6	1.97/6	1.66/5	1.72/5	1.74/5	1.84/5	1.95/5	2.00/5	2.09/5
	6	2.25/7	1.97/6	1.68/5	1.66/5	1.79/5	1.85/5	1.84/5	1.95/5	2.08/5	2.16/5
	7	2.36/7	2.07/6	1.77/5	1.79/5	1.85/5	1.93/5	1.99/5	2.07/5	2.08/5	2.24/5
	8	2.10/6	2.15/6	1.82/5	1.88/5	1.94/5	2.01/5	2.08/5	2.17/5	2.24/5	2.33/5
	9	2.22/6	2.15/6	1.82/5	1.96/5	2.02/5	2.08/5	2.17/5	2.17/5	2.32/5	2.41/5
	10	2.31/6	2.36/6	2.01/5	2.04/5	2.10/5	2.18/5	2.24/5	2.33/5	2.40/5	2.47/5



Fig. 4 Computational time and number of iterations of a geometric multigrid solver with the MS-BRB-GS(α) smoother

BRB-GS) smoother for geometric multigrid methods. It is a modified version of the block red-black Gauß–Seidel (BRB-GS) smoother that improves convergence rate and data locality by applying multiple consecutive Gauß–Seidel sweeps on each block.

The unknowns are divided into blocks so that the amount of data for processing each block fits into the cache, and α Gauß–Seidel iterations are applied to the block per smoother step. The computational cost for the additional iterations is much lower than for the first iteration because of data locality.

Figure 4 shows the effect of the MS-BRB-GS(α) smoother on a single node of the ITO system (Intel Xeon Gold 6154 (Skylake-SP) Cluster at Kyushu University). By increasing the number of both pre- and post-smoothing steps, the number of iterations is decreased. In the best case, MS-BRB-GS is 1.64× faster than BRB-GS.

3.2 The BEAST Framework for Interior Definite Generalized Eigenproblems

The BEAST framework targets the solution of interior definite eigenproblems

$$AX = BX\Lambda \ ,$$

i.e., for finding all eigenvectors and eigenvalues of a definite matrix pair (A, B), with A and B Hermitian and B additionally positive definite, within a given interval $[\underline{\lambda}, \overline{\lambda}]$. The framework is based on the Rayleigh–Ritz subspace iteration procedure, in particular the spectral filtering approach: Arbitrary continuous portions of the spectrum may be selected for computation with appropriate filtering functions that are applied via an implicit approximate projector to compute a suitable subspace

basis. Starting with an initial subspace Y, the following three main steps are repeated until a suitable convergence criterion is met:

Compute a subspace U by approximately projecting Y
Rayleigh–Ritz extraction: solve the reduced eigenproblem $A_U V = B_U V \Lambda$, where $A_U = U^H A U$, $B_U = U^H B U$, and let $X = UV$
Obtain new Y from X or U

In the following we highlight some of BEAST's algorithmic features, skipping other topics such as locking converged eigenpairs, adjusting the dimension of the subspace, and others.

3.2.1 Projector Types

BEAST provides three variants of approximate projectors. First, polynomial approximation (BEAST-P) using Chebyshev polynomials, which only requires matrix vector multiplications but is restricted to standard eigenproblems. Second, Cauchy integral-based contour integration (BEAST-C), as in the FEAST method [63]. As a third method, an iterative implementation of the Sakurai–Sugiura method [65] is available (BEAST-M), which shares algorithmic similarities with FEAST. In the following we briefly elaborate on the algorithmic ideas.

- In BEAST-P, we have $U = p(A) \cdot Y$ with a polynomial $p(z) = \sum_{k=0}^{d} c_k T_k(z)$ of suitable degree d. Here, T_k denotes the kth Chebyshev polynomial,

$$T_0(z) \equiv 1, \quad T_1(z) = z, \quad T_k(z) = 2z \cdot T_{k-1}(z) - T_{k-2}(z), \ k \geq 2.$$

  Due to the use of the T_k, this method is also known as Chebyshev filter diagonalization.
  In addition to well-known methods for computing the coefficients c_k [18, 62], BEAST also provides the option of using new, improved coefficients [25]. Their computation depends on two parameters, μ and σ, and for suitable combinations of these, the filtering quality of the polynomial can be improved significantly; see Fig. 5, which shows the "gain," i.e., the reduction of the width of those λ values *outside* the search interval, for which a damping of corresponding eigenvectors by at least a factor 100 cannot be guaranteed. For some combinations (σ, μ), marked red in the picture, this "no guarantee" area can be reduced by a factor of more than 2, which in turn allows using lower-degree polynomials to achieve comparable overall convergence. A parallelized method for finding suitable parameter combinations and computing the c_k is included with BEAST.

www.dbooks.org

- In BEAST-C, the exact projection

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \, (zB - A)^{-1} BY$$

(integration is over a contour Γ in the complex plane that encloses the eigenvalues $\lambda \in [\underline{\lambda}, \overline{\lambda}]$, but no others) is approximated using an N-point quadrature rule,

$$U = \sum_{j=1}^{N} \omega_j (z_j B - A)^{-1} BY,$$

leading to N linear systems, where the number of right-hand sides (RHS) corresponds to the dimension of the current subspace U (and Y).
- BEAST-M is also based on contour integration, but moments are used to reduce the number of RHS in the linear systems. Taking M moments, we have

$$U = [U_0, \ldots, U_{M-1}] \quad \text{with} \quad U_k = \sum_{j=1}^{N} \omega_j z_j^k (z_j B - A)^{-1} BY,$$

and thus an M times smaller number of RHS (dimension of Y) is sufficient to achieve the same dimension of U.

The linear systems in the contour-based schemes may be ill-conditioned if the integration points z_j are close to the spectrum (this happens, e.g., for narrow search intervals $[\underline{\lambda}, \overline{\lambda}]$); cf. also Sects. 3.1 and 3.4 for approaches to address this issue.

3.2.2 Flexibility, Adaptivity and Auto-Tuning

The BEAST framework provides flexibility at the algorithmic, parameter, and working precision levels, which we describe in detail in the following.

Algorithmic Level

The projector can be chosen from the three types described above, and the type may even be changed between iterations. In particular, an innovative subspace-iterative version of Sakurai–Sugiura methods (SSM) has been investigated for possible cost savings in the solution of linear systems via a limited subspace size and the overall reduction of number of right hand sides over iterations by using moments. Given, however, the potentially reduced convergence threshold with a constrained subspace size, we support switching from the multi-moment method, BEAST-M, to a single-moment method, BEAST-C. The efficiency, robustness, and

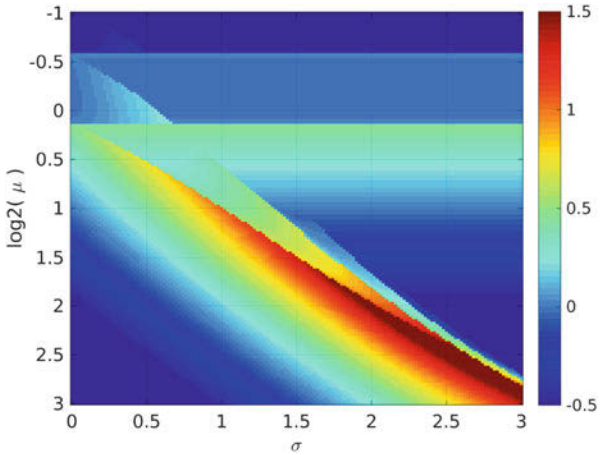


Fig. 5 Base-2 log of the "gain" from using modified coefficients with parameters (σ, μ) for the interval $[\underline{\lambda}, \overline{\lambda}] = [-0.584, -0.560]$ (matrix scaled such that $\text{spec}(A) = [-1, +1]$) and degree $d = 1600$

accuracy of this approach in comparison with traditional SSM and FEAST has been explored [35].

We further studied this scheme along with another performance-based implementation of SSM, z-PARES [65, 67]. These investigations considered the scaling and computational cost of the libraries as well as heuristics for parameter choice, in particular with respect to the number of quadrature nodes. We observed that the scaling behavior improved when the number of quadrature nodes increased, as seen in Fig. 6. As the linear systems solved at each quadrature node are independent and the quality of numerical integration improves with increased quadrature degree, exploiting this property makes sense, particularly within the context of exascale computations. However, it is a slightly surprising result, as previous experiments with FEAST showed diminishing returns for convergence with increased quadrature degree [23], something we do not observe here.

Parameter Level

In addition to the projector type, several algorithmic parameters determine the efficiency of the overall method, most notably the dimension of the subspace and the degree of the polynomial (BEAST-P) or the number of integration nodes (BEAST-C and BEAST-M).

With certain assumptions on the overall distribution of the eigenvalues, clear recommendations for optimum subspace size (as a multiple of the number of expected eigenvalues) and the degree can be given, in the sense that overall work

www.dbooks.org

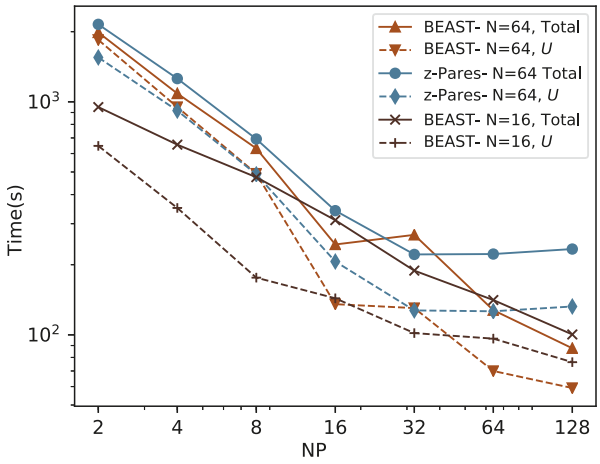


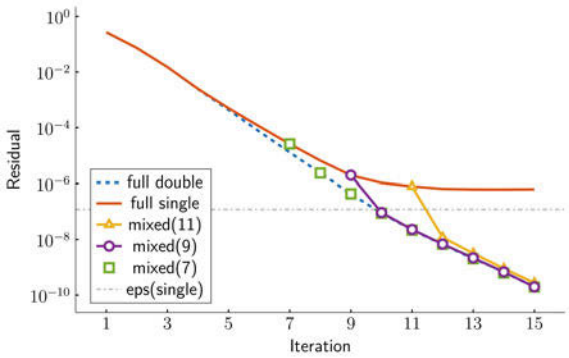
Fig. 6 Strong scaling of BEAST and z-Pares for a 1M × 1M standard eigenproblem based on a graphene sheet of dimension 2000 × 500. Both solvers found 260 eigenpairs in the interval $[-0.01, 0.01]$ to a tolerance of 1×10^{-8}. Both methods used 4 moments and began with random initial block vector Y. For BEAST, Y contained 100 columns; for z-Pares, 130. Testing performed on the Emmy HPC cluster at RRZE. MUMPS was used for the direct solution of all linear systems. N refers to the number of quadrature nodes along a circular contour, NP to the number of processes

is minimized. For more details, together with a description of a performance-tuned kernel for the evaluation of $p(A) \cdot Y$, the reader is referred to [62].

If such information is not available, or for the contour integration-type projectors, a heuristic has been developed that automatically adjusts the degree (or number of integration nodes) during successive iterations in order to achieve a damping of the unwanted components by a factor of 100 per iteration, which leads to close-to-optimum overall effort; cf. [26].

Working Precision Level

Given the iterative nature of BEAST, with one iteration being comparatively expensive, the possibility to reduce the cost of at least some of these iterations is attractive. We have observed that before a given residual tolerance is surpassed, systematic errors in the computation of the projector and other operations do not impair convergence speed per se, but impose a limit on what residual can be reached before progress stagnates. One such systematic error is the finite accuracy of floating-point computations, which typically are available in single and double precision. In the light of the aforementioned behavior, it seems natural to perform initial iterations in single precision and thereby save on computation time before a switch to double precision becomes inevitable; cf. Fig. 7.



Run / precision	Time total	Time projection
double	420 s	305 s
mixed(7)	333 s	265 s
mixed(9)	316 s	257 s
mixed(11)	301 s	245 s
single	266 s	225 s

Fig. 7 Left: average residual over the BEAST iterations for using double or single precision throughout, and for switching from single to double precision in the 7th, 9th, or 11th iteration, respectively. Right: time (in seconds) to convergence for a size 1,048,576 topological insulator (complex values) with a search space size of 256 and a polynomial degree of 135 on 8 nodes of the Emmy-cluster at RRZE. Convergence is reached after identical numbers of iterations (with the exception of pure single precision, of course). The timings can vary for different ratios of polynomial degree and search space size and depend on the single precision performance of the underlying libraries

Therefore, mixed precision has been implemented in all BEAST schemes mentioned above, allowing an adaptive strategy to automatically switch from single to double precision after a given residual tolerance is reached. A comprehensive description and results are presented in [4]. These results and our initial investigations also suggest that increased precision beyond double precision (i.e., quad precision) will have no benefit for the convergence rate until a certain double precision specific threshold is reached; convergence beyond this point would require all operations to be carried out with increased precision.

3.2.3 Levels of Parallelism

The BEAST framework exploits multiple levels of parallelism using an MPI+X paradigm. We rely on the GHOST and PHIST libraries for efficient sparse matrix/dense vector storage and computation; cf. Sect. 5. The operations implemented therein are themselves hybrid parallel and constitute the lowest level of parallelism in BEAST. Additional levels are addressed by parallelizing over blocks of vectors in Y and, for BEAST-C and BEAST-M, over integration nodes during the application of the approximate projector. A final level is added by exploiting the ability of the method to subdivide the search interval $[\underline{\lambda}, \overline{\lambda}]$ and to process the subintervals independently and in parallel. Making use of these properties, however, may lead to non-orthogonal eigenvectors, which necessitates postprocessing as explained in the following.

www.dbooks.org

3.2.4 A Posteriori Cross-Interval Orthogonalization

Rayleigh–Ritz-based subspace iteration algorithms naturally produce a B-orthogonal set of eigenvectors X, i.e., $\mathrm{orth}(X)$ is small, where

$$\mathrm{orth}(X) = \max\left\{\mathrm{orth}(x_i, x_j) | i \neq j\right\} \quad \text{with} \quad \mathrm{orth}(x, y) = \frac{\langle y, x\rangle}{\|x\|\|y\|}.$$

By contrast, the orthogonality

$$\mathrm{orth}(X, Y) = \max\left\{\mathrm{orth}(x_i, y_j)\right\}$$

between two or more independently computed sets of eigenvectors may suffer if the distance between the involved eigenvalues is small [49, 50]. Simultaneous re-orthogonalization of evolving approximate eigenvectors during subspace iteration has proven ineffective unless the vectors have advanced reasonably far. A large scale re-orthogonalization of finished eigenvector blocks, on the other hand, requires a careful choice of methodology in order to not diminish the quality of the previously established residual.

Orthogonalization of multiple vector blocks implies Gram–Schmidt style propagation of orthogonality, assuming $\mathrm{orth}(X, Y)$ can be arbitrarily poor. In practice, the independently computed eigenvectors will exhibit multiple grades of orthogonality, but rarely will there be no orthogonality (in the sense above) at all. This, in turn, allows for the use of less strict orthogonalization methods. While, in theory, the orthogonalization of p blocks requires at least $p(p-1)/2 + (p-1)$ block-block or intra-block orthogonalizations and ensures global orthogonality, an iterative scheme allows for more educated choices on the ordering of orthogonalizations in order to reduce losses in residual and improve the communication pattern, eliminating the need for broadcasts of vector blocks at the cost of additional orthogonalization operations in the form of multiple sweeps. In practice, very few sweeps ($\sim$2) are sufficient in most cases.

Every block-block orthogonalization $X = X - Y(Y^H BX)$ disturbs the orthogonality $\mathrm{orth}(X)$ of the modified block, as well as its residual. Local re-orthogonalization of X disturbs the residual further. We have identified orthogonalization patterns and selected orthogonalization algorithms that reduce the loss of residual accuracy to a degree that essentially eliminates the need for additional post-iteration.

The implementation of an all-to-all interaction of many participating vector blocks can be performed in multiple ways with different requirements regarding storage, communication, runtime, and with different implications on accuracy and loss of residual. Among several such strategies and algorithms that have been implemented and tested, the most promising is a purely iterative scheme, both for global and local orthogonalization operations. It is based on a comparison of interval properties, most notably the achieved residual from the subspace iteration. We are continuing to explore the possibility to detect certain orthogonalizations as

unnecessary without computing the associated inner products in order to further reduce the workload without sacrificing orthogonality.

3.2.5 Robustness and Resilience

In the advent of large scale HPC clusters, hardware faults, both detectable an undetectable, have to be expected.

Detectable hardware faults, e.g., the outage of a component that violently halts execution, can typically only be mitigated by frequent on-the-fly storage of the most vital information. In the case of subspace iteration, as is used in BEAST, almost all required information for being able to resume computation is encoded in the iterated subspace basis in form of the approximate eigenvectors, besides runtime information about the general program flow. Relying on the CRAFT library [73], a per-iteration checkpointing mechanism has been implemented in BEAST.

Additionally, for also being able to react to "silent" computation errors that merely distort the results but do not halt execution, the most expensive operation (application of the approximate projector) has been augmented to monitor the sanity of the results. This can be done in two ways: A checksum-style entrainment of additional vectors, linear combinations of the right-hand sides, can be checked during and after the application of the projector to detect errors and allow for the re-computation of the incorrect parts. The comparison of approximate filter values obtained from the computed basis and the expected values obtained from the scalar representation of the filter function, on the other hand, gives an additional a posteriori test for the overall plausibility of the basis.

Practical tests have shown that small distortions of the subspace basis have not enough impact on the overall process in order to justify expensive measures. If the error is not recurring, just continuing the subspace iteration is often the best and most cost-efficient option. This is particularly true in early iterations, where small errors have no effect at all.

3.3 Further Progress on Contour Integral-Based Eigensolvers

3.3.1 Relationship Among Contour Integral-Based Eigensolvers

The complex moment-based eigensolvers such as the Sakurai–Sugiura method can be regarded as projection methods using a subspace constructed by the contour integral

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \; z^k (zB - A)^{-1} BY.$$

www.dbooks.org

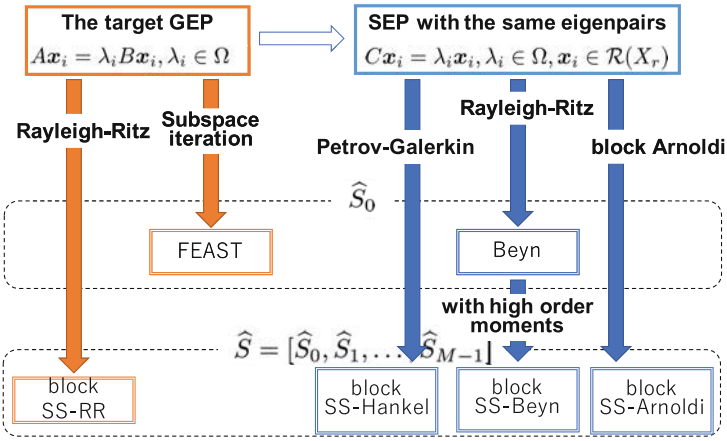


Fig. 8 A map of the relationships among the contour integral-based eigensolvers

The property of the subspace is well analyzed by using a filter function

$$f(\lambda) := \sum_{j=1}^{d} \frac{\omega_j}{z_j - \lambda},$$

which approximates a band-pass filter for the target region where the wanted eigenvalues are located. Using the filter function, error analyses of the complex moment-based eigensolvers were shown in [30, 41, 42, 66, 76]. By using the results of the error analyses, an error resilience technique and an accuracy deterioration technique have also been given in [32, 43].

The relationship between typical complex moment-based eigensolvers was also analyzed in [42] focusing on the subspace. The block SS-RR method [36] and the FEAST algorithm [76] are projection methods for solving the target generalized eigenvalue problem, whereas the block SS-Hankel method [37], Beyn [12], the block SS-Arnoldi methods [40] and its improvements [38] are projection methods for solving an implicitly constructed standard eigenvalue problem; see [42] for details. Figure 8 shows a map of the relationships among the contour integral-based eigensolvers.

3.3.2 Extension to Nonlinear Eigenvalue Problems

The complex moment-based eigensolvers were extended to nonlinear eigenvalue problems (NEPs):

$$T(\lambda_i)x_i = 0, \quad x_i \in \mathbb{C}^n \setminus \{0\}, \quad \lambda_i \in \Omega \subset \mathbb{C},$$

where the matrix-valued function $T : \Omega \to \mathbb{C}^{n\times n}$ is holomorphic in an open domain Ω. The projection for a nonlinear matrix function $T(\lambda)$ is given by

$$\frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \mathrm{d}z \; z^k T(z)^{-1} Y.$$

This projection is approximated by

$$U_k = \sum_{j=1}^{N} \omega_j z_j^k T(z_j)^{-1} Y, \quad k = 0, 1, \ldots, m-1.$$

The block SS-Hankel [7, 8], block SS-RR [83], and block SS-CAA methods [39] are simple extensions of the GEP solvers. A technique for improving the numerical stability of the block SS-RR method for NEP was developed in [15, 16].

Beyn proposed a method using Keldysh's theorem and the singular value decomposition [12]. Van Barel and Kravanja proposed an improvement of the Beyn method using the canonical polyadic (CP) decomposition [10].

3.4 Recursive Algebraic Coloring Engine (RACE)

The standard approach to solve the ill-conditioned linear systems arising in BEAST-C or FEAST is to use direct solvers. However, in [24] it was shown that the Kaczmarz iterative solver accelerated by a Conjugate Gradient (CG) method (the so-called CGMN solver [29]) is a robust alternative to direct solvers. Standard multicoloring (MC) was used in [29] for the parallelization of the CGMN kernels. After analyzing the shortcomings of this strategy in view of hardware efficiency, we developed in collaboration with the EXASTEEL-II project the Recursive Algebraic Coloring Engine (RACE) [1]. It is an alternative to the well-known MC and algebraic block multicoloring (ABMC) algorithms [44], which have the problem that their matrix reordering can adversely affect data access locality. RACE aims at improving data locality, reducing synchronization, and generating sufficient parallelism while still retaining simple matrix storage formats such as compressed row storage (CRS). We further identified distance-2 coloring of the underlying graph as an opportunity for parallelization of the symmetric spMVM (SymmSpMV) kernel.

RACE is a sequential, recursive, level-based algorithm that is applicable to general distance-k dependencies. It is currently limited to matrices with symmetric structure (undirected graph), but possibly nonsymmetric entries. The algorithm comprises four steps: level construction, permutation, distance-k coloring, and load balancing. If these steps do not generate sufficient parallelism, recursion on subgraphs can be applied. Using RACE implies a pre-processing and a processing phase. In pre-processing, the user supplies the matrix, the kernel requirements (e.g., distance-1 or distance-2) and hardware settings (number of threads, affinity

www.dbooks.org

strategy). The library generates a permutation and stores the recursive coloring information in a level tree. It also creates a pool of pinned threads to be used later. In the processing phase, the user provides a sequential kernel function which the library executes in parallel as a callback using the thread pool.

Figure 9 shows the performance of SymmSpMV on a 24-core Intel Xeon Skylake CPU for a range of sparse symmetric matrices. In Fig. 9a we compare RACE against Intel's implementation in the MKL library, and with roofline limits obtained via bandwidth measurements using array copy and read-only kernels, respectively. RACE outperforms MKL by far. In Fig. 9b we compare against standard multicoloring (MC) and algebraic block multicoloring (ABMC). The advantage of RACE is especially pronounced with large matrices, where data traffic and locality of access is pivotal. One has to be aware that some algorithms may exhibit a change in convergence behavior due to the reordering. This has to be taken into account when benchmarking whole program performance instead of kernels. Details can be found in [1].

In order to show the advantages of RACE in the context of a relevant algorithm, we chose FEAST [63] for computing inner eigenvalues. The hot spot of the algorithm (more than 95%) is a solver for shifted linear systems ($A-\sigma I = b$). These systems are, however, highly ill-conditioned, posing severe convergence problems for most linear iterative solvers. We use the FEAST implementation of Intel MKL, which by default employs the PARDISO direct solver [69], but its Reverse Communication Interface (RCI) allows us to plug our CGMN implementation instead. In the following experiment we find ten inner eigenvalues of a simple discrete Laplacian matrix to an accuracy of 10^{-8}. Figure 10 shows the measured time and memory footprint of the default MKL version (using PARDISO) and the CGMN versions parallelized using both RACE and ABMC for different matrix sizes. ABMC is a factor of $4\times$ slower than RACE. The time required by the default MKL with PARDISO is smaller than with CGMN using RACE for small sizes; however, the gap gets smaller as the size grows due to the direct solvers having a higher time complexity (here $\approx O(n^2)$) compared to iterative methods ($\approx O(n^{1.56})$). Moreover, the direct solver requires more memory, and the memory requirement grows much faster (see Fig. 10b) than with CGMN. In our experiment the direct solver ran out of memory at problem sizes beyond 140^3, while CGMN using RACE used less than 10% of space at this point. Thus, CGMN with RACE can solve much larger problems compared to direct solvers, which is a major advantage in fields like quantum physics.

4 Hardware Efficiency and Scalability

In this section we showcase performance and parallel efficiency numbers for library components developed in ESSEX-II that are of paramount importance for the application work packages: GPGPU-based tall and skinny matrix-matrix multiplication and the computation of inner eigenvalues using polynomial filter techniques.

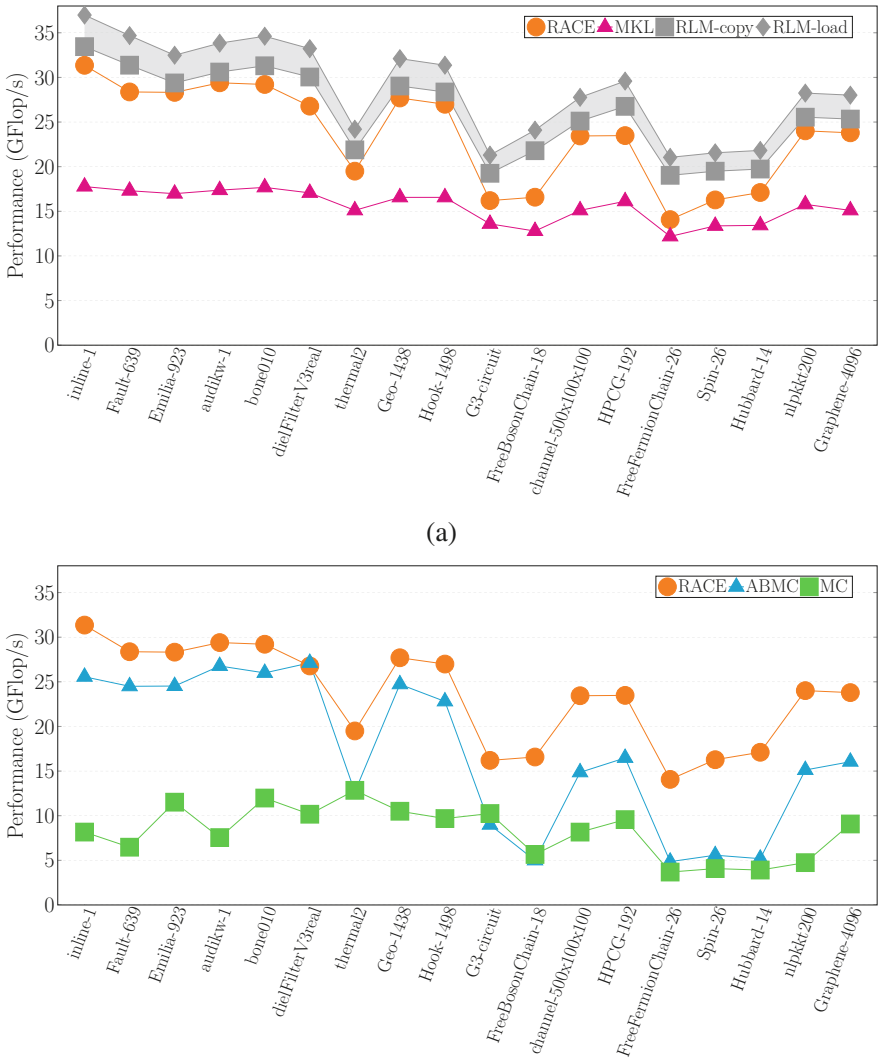


Fig. 9 SymmSpMV performance of RACE compared to other methods. The roofline model for SymmSpMV is shown in Fig. 9a for reference. Representative matrices from [17] and ScaMaC (see Sect. 5.5) were used. Note that the matrices are ordered according to increasing number of rows. (One Skylake Platinum 8160 CPU [24 threads]). (**a**) Performance of RACE compared with MKL. (**b**) Performance of RACE compared to other coloring approaches

www.dbooks.org

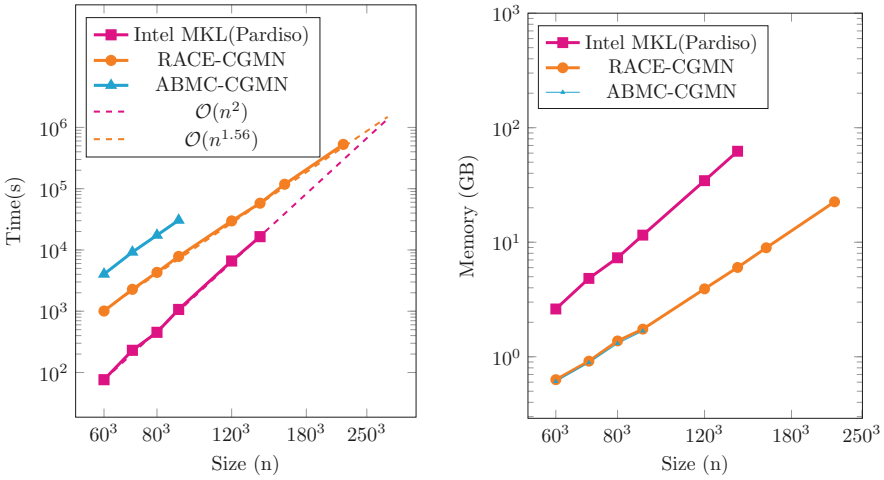


Fig. 10 Comparison of FEAST with default MKL direct solver and iterative solver CGMN, parallelized using RACE. (One Skylake Platinum 8160 CPU [24 threads]). (**a**) Time to solution. (**b**) Memory requirement

4.1 Tall and Skinny Matrix-Matrix Multiplication (TSMM) on GPGPUs

Orthogonalization algorithms frequently require the multiplication of matrices that are strongly nonsquare. Vendor-supplied optimized BLAS libraries often yield sub-optimal performance in this case. "Sub-optimal" is a well-defined term here since the multiplication of an $M \times K$ matrix A with a $K \times N$ matrix B with $K \gg M, N$ and small M, N is a memory-bound operation: At $M = N$, its computational intensity is just $M/8$ flop/byte. In ESSEX-I, efficient implementations of TSMM on multicore CPUs were developed [51].

The naive roofline model predicts memory-bound execution for $M \lesssim 64$ on a modern Volta-class GPGPU. See Fig. 11 for a comparison of optimal (roofline) performance and measured performance for TSMM on an Nvidia Tesla V100 GPGPU using the cuBLAS library.[6] We have developed an implementation of TSMM for GPGPUs [20], investigating various optimization techniques such as different thread mappings, overlapping long-latency loads with computation via leapfrogging[7] and unrolling, options for global reductions, and register tiling. Due

[6] https://docs.nvidia.com/cuda/cublas (May 2019).

[7] Leapfrogging in this context means that memory loads to operands are initiated one loop iteration before the data is actually needed, allowing for improved overlap between data transfers and computations.

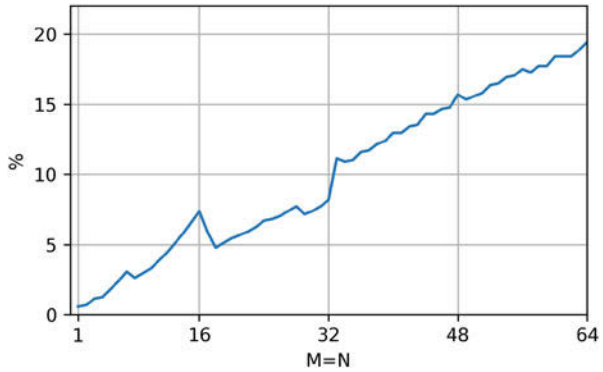


Fig. 11 Percentage of roofline predicted performance achieved by cuBLAS for the range $M = N \in [1, 64]$ on a Tesla V100 with 16 GB of memory. (From [20])

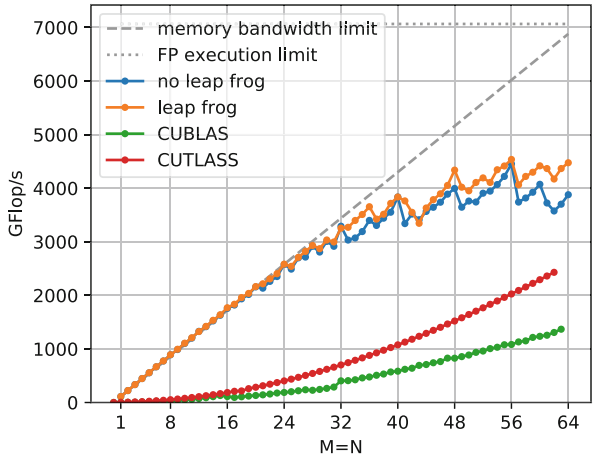


Fig. 12 Best achieved performance for each matrix size with $M = N$ in comparison with the roofline limit, cuBLAS and CUTLASS, with $K = 2^{23}$. (From [20])

to the large and multi-dimensional parameter space, the kernel code is generated using a python script.

Figure 12 shows a comparison between our best implementations obtained via parameter search (labeled "leap frog" and "no leap frog," respectively) with cuBLAS and CUTLASS,[8] which is a collection of CUDA C++ template abstractions for high-performance matrix multiplications. Up to $M = N = 36$, our implementation stays within 95% of the bandwidth limit. Although the performance

[8] https://github.com/NVIDIA/cutlass (May 2019).

www.dbooks.org

levels off at larger M, N, which is due to insufficient memory parallelism, it is still significantly better than with cuBLAS or CUTLASS.

4.2 BEAST Performance and Scalability on Modern Hardware

4.2.1 Node-Level Performance

Single-device benchmark tests for BEAST-P were performed on an Intel Knights Landing (KNL), an Nvidia Tesla P100, and an Nvidia Tesla V100 accelerator, comparing implementations based on vendor libraries (MKL and cuBLAS/cuSPARSE, respectively) with two versions based on GHOST: one with and one without tailored fused kernels. The GPGPUs showed performance levels expected from a bandwidth-limited code, while on KNL the bottleneck was located in the core (see Fig. 13a). Overall, the concept of fused optimized kernels provided speedups of up to 2× compared to baseline versions. Details can be found in [56].

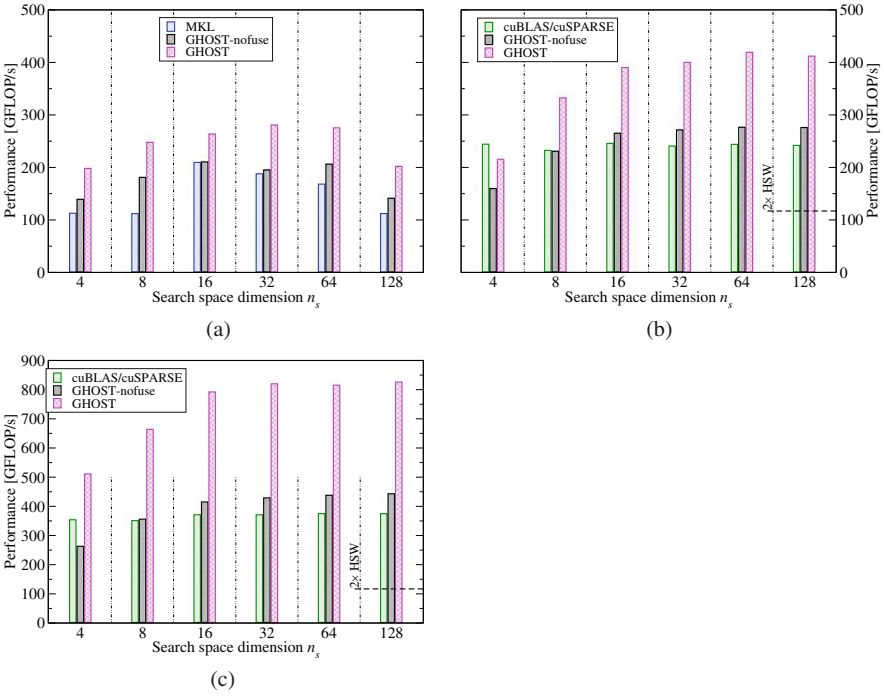


Fig. 13 BEAST-P performance for a topological insulator problem of dimensions 128×64×64 with $n_p = 500$ using different implementations on KNL, P100, and V100. Performance of a dual Xeon E5-2697v3 node (Haswell) is shown for reference. Note the different y axis scaling of the V100 results. (From [56]; for details see therein). (**a**) KNL. (**b**) P100. (**c**) V100

4.2.2 Massively Parallel Performance

Scaling tests for BEAST-P were performed on the "Oakforest-PACS" (OFP) at the University of Tokyo, "Piz Daint" at CSCS in Lugano, and on the "SuperMUC-NG" (SNG) at Leibniz Supercomputing Centre (LRZ) in Garching.[9] While the OFP nodes comprise Intel "Knights Landing" (KNL) many-core CPUs, SNG has CPU-only dual-socket nodes with Intel Skylake-SP, and Piz Daint is equipped with single-socket Xeon "Haswell" nodes, each of which has an Nvidia Tesla P100 accelerator attached. Weak and strong scaling tests were done with topological insulator (TI) matrices generated by the ScaMaC library. Flops were calculated for the computation of the approximate eigenspace, U, averaged over the four BEAST iterations it took to find the 148 eigenvalues in each interval to a tolerance of 1×10^{-10}. The subspace contained 256 columns, and spMMVs were performed in blocks of size 32 for best performance. Optimized coefficients [25] were used for the Chebyshev polynomial approximation, resulting in a lower overall required polynomial degree. Weak and strong scaling results are shown in Fig. 14a through d.

OFP and SNG show similar weak scaling efficiency due to comparable single-node performance and network characteristics. Piz Daint, owing to its superior single-node performance of beyond 400 Gflop/s, achieves only 60% of parallel efficiency at 2048 nodes. A peculiar observation was made on the CPU-only SNG system: Although the code runs fastest with pure OpenMP on a single node (223 Gflop/s), scaled performance was observed to be better with one MPI process per socket. The ideal scaling and efficiency numbers in Fig. 14a–c use the best value on the smallest number of nodes in the set as a reference. The largest matrix on SNG had 6.6×10^9 rows.

5 Scalable and Sustainable Software

It was a central goal of the ESSEX-II project to consolidate our software efforts and provide a library of solvers for sparse eigenvalue problems on extreme-scale HPC systems. This section gives an overview of the status of our software, most of which is now publicly available under a three-clause BSD license. Many of the efforts have been integrated in the PHIST library so that they can easily be used together, and we made part of the software available in larger contexts like Spack [27] and the extreme-scale scientific software development kit xSDK [11]. The xSDK is an effort to define common standards for high-performance, scientific software in terms of software engineering and interoperability.

[9] Runs on OFP and SNG were made possible during the "Large-scale HPC Challenge" Project on OFP and the "Friendly-User Phase" of SNG.

www.dbooks.org

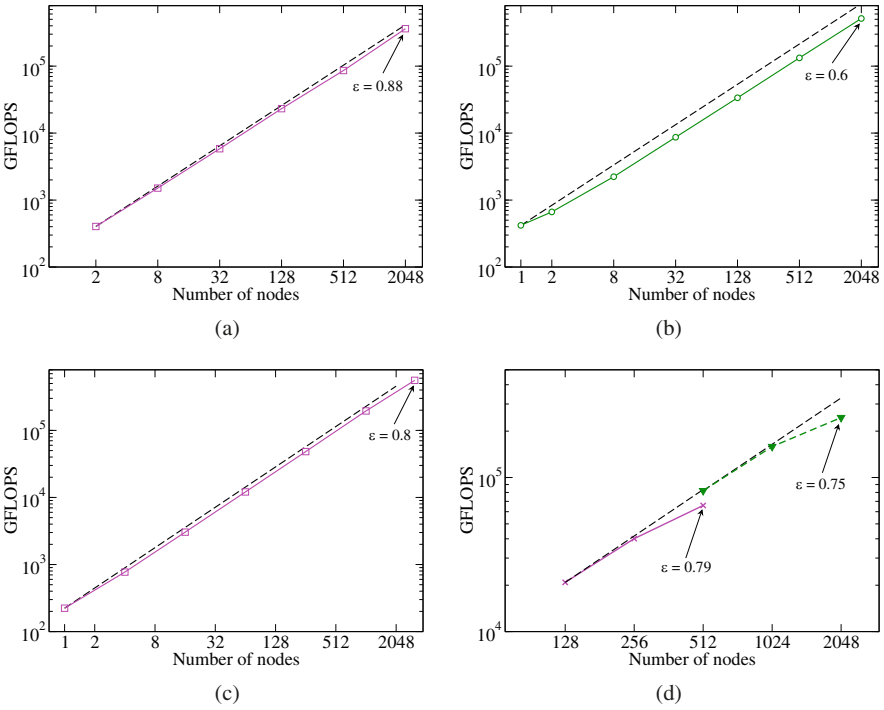


Fig. 14 Weak scaling of BEAST-P on OFP, Piz Daint, and SNG, and strong scaling on SNG. Dashed lines denote ideal scaling with respect to the smallest number of nodes in the set. (**a**) Weak scaling of BEAST-P on OFP for problems of size 2^{20} (2 nodes) to 2^{30} (2048 nodes, about one quarter of the full machine). (**b**) Weak scaling of BEAST-P on Piz Daint for problems of size 2^{21} (1 node) to 2^{32} (2048 nodes). (From [56]). (**c**) Weak scaling of BEAST-P on SNG for problems of size 2^{21} (1 node) to 1.53×2^{32} (3136 nodes, about half of the full machine). (**d**) Strong scaling of BEAST-P on SNG for problems of size 2^{28} (crosses) and 2^{30} (triangles)

The current status of the software developed in the ESSEX-II project is summarized as follows.

- BEAST is available via bitbucket,[10] and can be compiled either using the PHIST kernel interface or the GHOST library directly. The former allows using it with any backend supported by PHIST.
- CRAFT is available stand-alone[11] or (in a fixed version) as part of PHIST.
- ScaMaC is available stand-alone[12] or (in a fixed version) as part of PHIST.

[10] https://bitbucket.org/essex/beast/.

[11] https://bitbucket.org/essex/craft/.

[12] https://bitbucket.org/essex/matrixcollection/.

- GHOST is available via bitbucket.[13] The functionality which is required to provide the PHIST interface can be tested via PHIST. Achieving full (or even substantial) test coverage of the GHOST-functionality would require a very large number of tests (in addition to what the PHIST interface provides, GHOST allows mixing data types, and it uses automatic code generation, which leads to an exponentially growing number of possible code paths with every new kernel, supported processor and data type). It is, however, possible to create a basic GHOST installation via the Spack package manager (since March 2018, commit `bcde376`).
- PHIST is available via bitbucket[14] and Spack (since commit `2e4378b`). Furthermore, PHIST 1.7.5 is part of xSDK 0.4.0. The version distributed with the xSDK is restricted to use the Tpetra kernels to maximize the interoperability of the package.

5.1 PHIST and the Block-ILU

In ESSEX-I we addressed mostly node-level performance [77] on multi-core CPUs. The main publication of ESSEX-II concerning the PHIST library [78] presents performance results for the block Jacobi–Davidson QR (BJDQR) solver on various platforms, including recent CPUs, many-core processors and GPUs. It was also shown in this work that the block variant has a clear performance advantage over the single-vector algorithm in the strong scaling limit. The reason is that, while the number of matrix-vector multiplications increases with the block size (see also [64]), the total number of reductions decreases. In order to demonstrate the performance portability of PHIST, we show in Fig. 15 a weak scaling experiment on the recent SuperMUC-NG machine.

For the block size 4, we roughly match the performance it achieves in the memory-bounded HPCCG benchmark (207 TFlop/s),[15] but using only half of the machine. This gives a clear indication that our node-level performance engineering and multi-node implementation are highly successful: after all, we do not optimize for the specific operator application (a simple structured grid, 3D Laplace operator), which the HPCCG code does. On the other hand, we have an increased computational intensity for some of the operations due to the blocking, which increases the performance over a single-vector CG solver. The single-vector BJDQR solver achieves 98 TFlop/s on half of the machine.

[13] https://bitbucket.org/essex/ghost/.

[14] https://bitbucket.org/essex/phist.

[15] See https://www.top500.org/system/179566.

www.dbooks.org

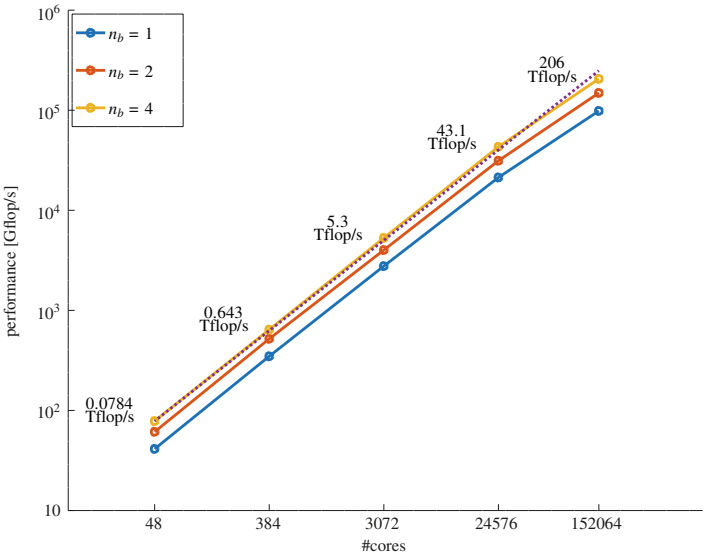


Fig. 15 Weak scaling behavior of the PHIST BJDQR solver for a symmetric PDE benchmark problem and different block sizes

5.1.1 Integration of the Block-ILU Preconditioning Technique

Initial steps have been taken to make the Block-ILU preconditioner (cf. Sect. 3.1) available via the PHIST preconditioning interface. At the time of writing, there is an experimental implementation of a block CRS sparse matrix format in the PHIST builtin kernel library, including parallel conversion and matrix-vector product routines and the possibility to construct and apply the block Cholesky preconditioner. Furthermore, the interfaces necessary to allow using the preconditioner within the BJDQR eigensolver have been implemented. These features are available for experimenting in a branch of the PHIST git repository because they do not yet meet the high demands on maintainability (especially unit testing) and documentation of a publicly available library. Integration of the method with the BEAST eigensolver is not yet possible because the builtin kernel library does not support complex arithmetic. As mentioned in Sect. 3.2, the complex version will be integrated directly into the BEAST software, instead.

5.2 BEAST

BEAST combines implementations of spectral filtering methods for Rayleigh–Ritz type subspace iteration in a generalized framework to provide facilities for improving performance and robustness. The algorithmic foundation allows for the

solution of interior Hermitian definite eigenproblems of standard and generalized form via an iterative eigensolver, unveiling all eigenpairs in one or many specified intervals. The software is designed as hybrid parallel library, written in C/C++, and relying on GHOST and PHIST to provide basic operations, parallelism, and data types. Beyond the excellent scalability of the underlying kernel libraries, multiple additional levels of parallelism allow for computing larger portions of the spectrum and/or utilizing a larger number of computing cores. The inherent ability of the underlying algorithm to compute separate intervals independently offers wide potential but requires careful handling of cross-interval interactions to ensure the desired quality of results, which is well supported by BEAST.

The BEAST library interface comes in variations for the common floating point formats (real and complex, single and double precision) for standard and generalized eigenproblems. Additionally, the software offers the possibility to switch precisions on-the-fly, from single to double precision, in order to further improve performance. While BEAST offers an algorithm for standard eigenproblems that completely bypasses the need for linear system solves, other setups typically require a suitable linear solver. Besides a builtin parallel sparse direct solver for banded systems, BEAST includes interfaces to MUMPS and Strumpack, as well as a flexible callback-driven interface for the inclusion of arbitrary linear solvers. It also interfaces with CRAFT and ScaMaC, which provide fault tolerance and dynamic matrix generation, respectively. While working out of the box for many problems, BEAST offers a vast amount of options to tweak the software for the specific problem at hand. A builtin command line parser allows for easy modification. The included application bundles the several capabilities of BEAST in form of a standalone tool that reads or generates matrices and solves the specified eigenproblem. As such, it acts as comprehensive example for the usage of BEAST.

The library is still in a development state, and interface and option sets may change. A more comprehensive overview over a selection of features is provided in Sect. 3.2.

5.3 CRAFT

The CRAFT library [73] covers two essential aspects of fault tolerance namely communication, and data recovery of an MPI application in case of process-failures.

In the Checkpoint/Restart part of the library, it provides an easier and extensible interface for making application-level checkpoint/restart. A CRAFT checkpoint can be defined simply by defining a `Checkpoint` object and adding the restart-relevant data in it, as shown in Listing 1. By default, the `Checkpoint::add()` function supports the most frequently used data formats, e.g., "plain old data" (POD), i.e., `int`, `double`, `float`, etc., POD 1D- and 2D-arrays, MPI data-types, etc.. However, it can be easily extended to support any user defined data-types. The `Checkpoint::read()`, `write()` and `update()` methods can then be used to read/write all added checkpoint's data. The library supports asynchronous-

www.dbooks.org

```
#include <mpi.h>
#include <craft.h>
int main(int argc, char* argv[]){
  ...
  size_t n=5, myrank, iteration=1, cpFreq=10;
  double dbl = 0.0;
  int * dataArr   = new int[n];
  MPI_Comm FT_Comm;
  MPI_Comm_dup (MPI_COMM_WORLD, &FT_Comm);
  AFT_BEGIN (FT_Comm, &myrank, argv);
  ********************************************
  Checkpoint  myCP ("myCP", FT_Comm);          * //define checkpoint
  myCP.add ("dbl", &dbl);                      *
  myCP.add ("iteration", &iteration);          *
  myCP.add ("dataArr", dataArr, &n);           AFT Zone
  myCP.commit();                               *
  myCP.restartIfNeeded (&iteration);           *
  for(; iteration <= 100 ; iteration++){       *
    Computation_communication();               *
    modifyData(&dbl, dataArr);                 *
    myCP.updateAndWrite (iteration, cpFreq);   *
  }                                            *
  ...                                          *
  ********************************************
  AFT_END();
}
```

Listing 1 A toy-code that demonstrates the simplicity of CRAFT's checkpoint/restart and automatic fault tolerace features in a typical iterative-style scientific application

checkpointing as well as node-level checkpointing using the SCR library [68]. Moreover it supports multi-staged, nested-, and signal-checkpointing.

The Automatic Fault Tolerance (AFT) part of CRAFT provides an easier interface for a dynamic process-failure recovery and management. CRAFT uses the ULFM-MPI implementation for process-failure detection, propagation, and communication recovery procedures, however it considerably reduces the user's effort by hiding these details behind `AFT_BEGIN()` and `AFT_END()` functions as shown in Listing 1. After a process failure, the library recovers the broken communicator (shrinking or non-shrinking by process-spawning), and returns the control back to the program at `AFT_BEGIN()`, where the data can be recovered. Both of these CRAFT functionalities are designed to complement each other, however they can be used independently as well. For detailed explanation of the features included in CRAFT, check [73]. Moreover, the library is available at [72].

5.4 CRAFT Benchmark Application

Within the scope of ESSEX, we have integrated CRAFT in the GHOST and PHIST libraries, and the BEAST algorithm.

Figure 16 shows a benchmark comparing the overhead of three different checkpointing strategies for the Lanczos algorithm (GHOST-based eigensolver), Jacobi–Davidson (PHIST-based eigensolver), and the BEAST algorithm. The important

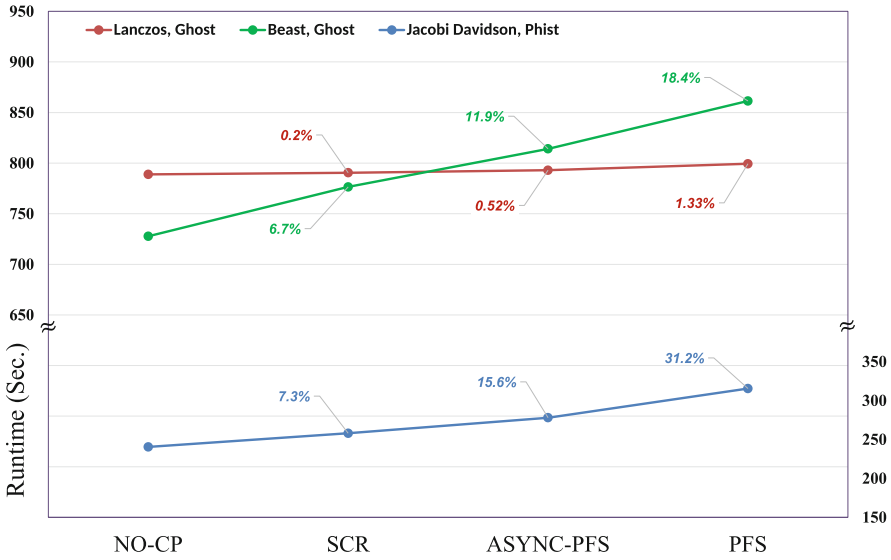


Fig. 16 CRAFT checkpointing overhead comparison for the Lanczos, Jacobi–Davidson, and BEAST eigenvalue solvers using three checkpointing methods of CRAFT, namely, node-level checkpointing with SCR, asynchronous PFS, and synchronous PFS checkpoints. The overhead for each checkpoint case is shown as a percentage. (Number of nodes=128, number of processes=256, Intel MPI)

parameters for these benchmarks are listed in Table 1. The benchmark shows that the node-level and asynchronous checkpointing significantly reduces the checkpoint overhead despite a very high checkpoint frequency.

The benchmark presented in Fig. 17 demonstrates the overhead caused by checkpoint/restart as well as by the communication recovery after process failures for the Lanczos application. The first two bars, namely 'No CP Intel MPI' and 'No CP ULFM-MPI' show the runtime between non-fault-tolerant (Intel-MPI) vs. a fault-tolerant MPI implementation (ULFM-MPI), and creates a baseline for ULFM-MPI implementation without any failures. The next two groups of bars show the application runtime with 0-,1-, and 2-failures with checkpoints taken on PFS- and node-level. The failures are triggered at the mid-point of two successive checkpoints from within the application to have a deterministic re-computation time, where each failure simulates a complete node-crash (2 simultaneous process failures) and recovery is performed in a non-shrinking fashion on spare nodes. The largest contribution to the overhead is caused by the re-computation part, whereas the communication repair overhead takes an average of ≈ 2.6 s only.

Besides ESSEX, CRAFT has been utilized in [22] to create a process-level fault tolerant FEM code based on the shrinking recovery style. Moreover, CRAFT has been recently integrated in the EXASTEEL [21] project.

www.dbooks.org

Table 1 The parameter values for Lanczos, JD, and Beast benchmarks

Lanczos parameters			
Matrix	Graphene-3000-3000	Number of rows and columns	$9.0 \cdot 10^8$
Number of non-zeros	$11.7 \cdot 10^9$	Global checkpoint size	≈14.4 GB
Number of iterations	3000	Checkpoint frequency	500
Jacobi–Davidson parameters (using Phist)			
Matrix	spinSZ30	Number of rows and columns	$1.6 \cdot 10^8$
Number of non-zeros	$2.6 \cdot 10^9$	Number of sought eigenvalues	20
Number of checkpoints	10		
Global checkpoint size	≈32 GB	Backend support library	GHOST
Beast parameters			
Matrix	tgraphene: 12000, 12000, 0	Number rows and columns	$1.44 \cdot 10^8$
Beast iterations	9	Checkpoint frequency	2
Global checkpoint size	≈65 GB	Backend support library	GHOST

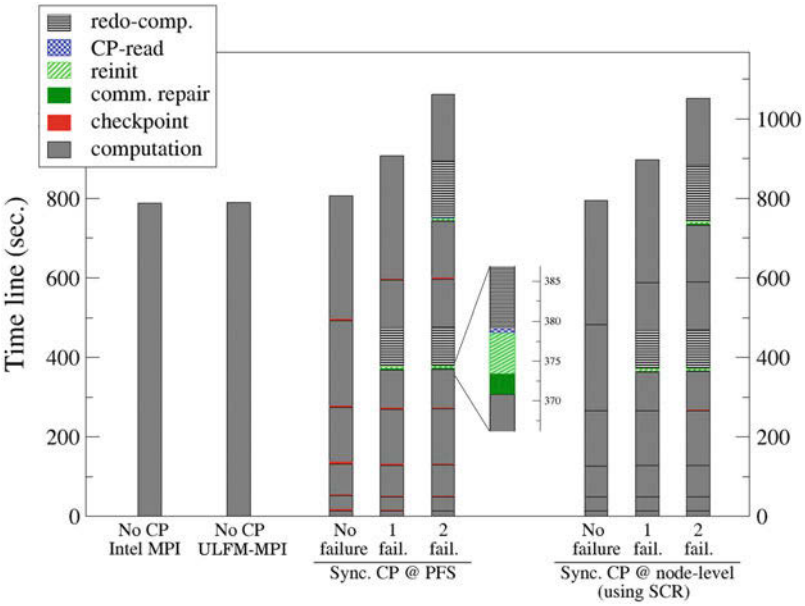


Fig. 17 Lanczos application with various checkpoint/restart and process failure recovery scenarios using 128 nodes (256 processes) on the RRZE Emmy cluster. On average the communication recovery time is 2.6 s (ULFM-MPI v1.1)

5.5 ScaMaC

Sparse matrices are central objects in the ESSEX project because of its focus on large-scale numerical linear algebra problems. A sparse matrix, whether derived from the Hamiltonian of a quantum mechanical system, from the Laplacian in a partial differential equation, or simply given as an abstract entity with unknown properties, defines a problem to be solved. The solution may then consist of a set of eigenvalues and eigenvectors computed with the BEAST or Jacobi–Davidson algorithms or, more moderately, of an estimate of some matrix norm or the spectral radius.

Testing and benchmarking of linear algebra algorithms, but also of computational kernels such as spMVM, requires matrices of different type and different size. Standard collections such as the Matrix Market [58] or Florida Sparse Matrix Collection [17] cover a wide range of examples, but mainly provide matrices of fixed moderate size. As algorithms and implementations improve, such matrices become readily too small and limited to serve as realistic test and benchmark cases.

We therefore decided in the ESSEX project to establish a collection of *scalable* matrices—the ScaMaC. Every matrix in ScaMaC is parameterized by individual parameters that allow the user to scale up the matrix dimension and to modify other, for example spectral, properties of the matrix. ScaMaC includes simple test and benchmark matrices but also 'real-world' matrices from research studies and applications. A major goal of ScaMaC is to provide a flexible yet generic interface for matrix generation, together with the necessary infrastructure to allow for immediate access to the collection irrespective of the concrete usage case.

The ScaMaC approach to matrix generation is straightforward and simple: Matrices are generated row-by-row (or column-by-column). The entire complexity of the actual generation technique, which depends on the specific matrix example, is encapsulated in a `ScamacGenerator` type and hidden from the user. ScaMaC provides routines to create and destroy such a matrix generator, to query matrix parameters prior to the actual matrix generation, and to obtain each row of the matrix. The ScaMaC interface is entirely generic and identical for all matrices in the collection.

A minimal code example is given in Fig. 18. In this example, the matrix and its parameters are set by parsing an argument string of the form `"MatrixName, parameter=...,..."` in line 3, before all rows are generated in the loop in lines 12–17. As this examples shows, parallelization of matrix generation is not part of the ScaMaC, but lies within the responsibility of the calling program. All ScaMaC routines are thread-safe and can be embedded directly into MPI processes and OpenMP threads. This approach guarantees full flexibility for the user and is easily integrated into existing parallel matrix frameworks such as PETSc or Trilinos. Both BEAST and PHIST provide direct access to the ScaMaC, therefore freeing the user from any additional considerations when using ESSEX software.

www.dbooks.org

```c
// step 1: obtain a generator - per process
ScamacGenerator * my_gen;
err = scamac_parse_argstr("Hubbard,n_sites=20", &my_gen, &errstr);
err = scamac_generator_finalize(my_gen);
    ................
// step 2: allocate workspace - per thread
ScamacWorkspace * my_ws;
err = scamac_workspace_alloc(my_gen, &my_ws);
    ................
// step 3: generate the matrix row by row
ScamacIdx nrow = scamac_generator_query_nrow(my_gen);
for (idx=0; idx<nrow; idx++) { // parallelize loop with OpenMP, MPI, ...
    // obtain the column indices and values of one row
    err = scamac_generate_row(my_gen, my_ws, idx, SCAMAC_DEFAULT, &nz, cind, val);
    // store or process the row
        ................
}
// step 4: clean up
err = scamac_workspace_free(my_ws);     // in each thread
err = scamac_generator_destroy(my_gen); // in each process
// step 5: use matrix
    ................
```

Fig. 18 Code example for row-by-row matrix generation with the generic ScaMaC generators

ScaMaC is written in plain C. Auto-generated code is included already in the release, such that requirements at compile time are minimal. Interoperability with other programming languages is straightforward, e.g., by using the ISO C bindings of the FORTRAN 2003 standard. Runtime requirements are equally minimal. Matrix generation has negligible memory overhead, requiring only a few KiB workspace to store lookup tables and similar information.

The key feature of ScaMaC is scalability, since the matrix rows (or columns) can be generated independently and in arbitrary order. For example at Oakforest-PACS (see Sect. 4.2.2), a `Hubbard` matrix (see below) with dimension $\geq 9 \times 10^9$ and $\geq 1.5 \times 10^{11}$ non-zeros is generated in less than a minute, using 2^{10} MPI processes each of which generates an average of 1.5×10^5 rows per second. As explained, the task of efficiently storing or using the matrix is left to the calling program.

ScaMaC is accompanied by a small toolkit for exploration of the collection. The toolkit addresses some basic tasks such as querying matrix information or plotting the sparsity pattern, but is not intended to compete with production-level code or full-fledged solver libraries, which the ESSEX project provides with BEAST, GHOST, and PHIST.

At the moment,[16] the matrix generators included in ScaMaC strongly reflect our personal research interests in quantum physics, but the ScaMaC framework is entirely flexible and allows for easy inclusion of new types of matrices, provided that they can be generated in a scalable way. The next update (scheduled for 2020) will extend ScaMaC primarily with matrix generators for standard partial differ-

[16] In version 0.8.2, ScaMaC contains 15 different matrix generators with a total of 95 parameters.

ential equations, including stencil matrices and finite element discretizations for advection-diffusion and elasticity problems, wave propagation, and the Schrödinger equation. Additional examples that are well suited for scalable generation are regular and irregular graph Laplacians, which have gained renewed interest in the context of machine learning [14, 59].

To obtain an idea of the 'real-world' application matrices already contained in ScaMaC, consider two examples: The celebrated Hubbard model of condensed matter physics (`Hubbard`) [34] and a theoretical model for excitons in the cuprous oxide from our own research in this field (`Exciton`) [2]. These matrices appear as Hamiltonians in the Schrödinger equation, and thus are either symmetric real (`Hubbard`) or Hermitian complex (`Exciton`). The respective application requires a moderate number (typically, 10–1000) of extremal or interior eigenpairs, which is less than 0.1% of the spectrum. Other ScaMaC generators provide general (non-symmetric or non-Hermitian) matrices, with a variety of sparsity patterns, spectral properties, etc. All generators depend on a number of application-specific parameters,[17] which are partly listed in Table 2 for the `Hubbard` and `Exciton` generator.

For the `Hubbard` example, two parameters determine the matrix dimension and sparsity pattern: n_fermions gives the number electrons per spin orientation (up or down), n_sites the number of orbitals occupied by the electrons. In terms of these parameters, the matrix dimension is $D = \binom{\text{n_sites}}{\text{n_fermions}}^2$. This dependency results in the rapid growth of D shown in Table 3. In the physically very interesting case of half-filling (n_fermions = n_sites/2 = n) we have asymptotically $D \simeq 2^n/\sqrt{(\pi/2)n}$, that is, exponential growth of D.

The `Exciton` example has the more moderate dependence $D = 3(2\mathrm{L}+1)^3$ (see Table 3). Here, the parameter L is a geometric cutoff that limits the maximal distance between the electron and hole that constitute the exciton. This example has a number of other parameters that are adapted literally from [2]. These parameters enter into the matrix entries, and thus affect the matrix spectrum and, finally, the algorithmic hardness of computing the eigenvalues of interest that determine the physical properties of the exciton.

Both `Hubbard` and `Exciton` are examples of difficult matrices, albeit for different reasons. For `Hubbard`, one unresolved challenge is to compute multiple interior eigenvalues for large n_fermions, n_sites, which becomes extremely difficult because of the rapid growth of the matrix dimension (specialized techniques for the Hubbard model such as the density-matrix renormalization group [70] cannot compute interior eigenvalues). Due to the irregular sparsity pattern of the `Hubbard` matrices (see Fig. 19 below), already the communication overhead of spMVM poses a serious obstacle to scalability and parallel efficiency. For `Exciton`, which are essentially stencil-like matrices of moderate size, the challenge is to compute some hundred eigenvalues out of a strongly clustered spectrum. Here, it is the

[17] For a full list of generators and parameters, consult the ScaMaC documentation included with the code, or at https://alvbit.bitbucket.io/scamac_docs/_matrices_page.html.

www.dbooks.org

Table 2 Parameters of the `Hubbard` and `Exciton` matrix generator in the ScaMaC

`Hubbard`		
matrix type: symmetric real		
int	n_sites	number of sites
int	n_fermions	number of fermions
double	t	hopping strength
double	U	Hubbard interaction
	⋮	
double	ranpot	random potential
rngseed	seed	random seed

`Exciton`		
matrix type: Hermitian complex		
int	L	cube length
double	so	spin orbit
double	ex	exchange
double	mlh	mass light hole
double	mhh	mass heavy hole
double	me	mass electron
double	eps	dielectric constant
	⋮	

Table 3 Matrix dimension D for the `Hubbard` and `Exciton` example, as a function of the respective parameter n_sites (and default value n_fermions = 5) or L

`Hubbard`	
n_sites	D
10	63 504
15	9 018 009
20	240 374 016
25	2 822 796 900
30	20 307 960 036
40	432 974 528 064

`Exciton`	
L	D
10	27 783
20	206 763
50	3 090 903
100	24 361 803
150	81 812 703
200	193 443 603

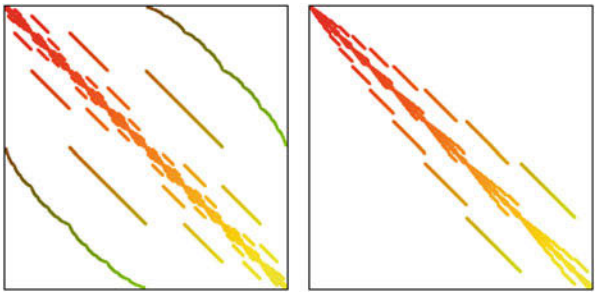


Fig. 19 Sparsity pattern of the Hubbard (`Hubbard,n_sites=40,n_fermions=20`) and spin chain (`SpinChainXXZ,n_sites=32,n_up=8`) example

poor convergence of iterative eigenvalue solvers for nearly degenerate eigenvalues that renders this problem hard. Thanks to the algorithmic advances in the ESSEX project, we now have reached a position that allows for future progress on these problems.

ScaMaC comes with several convenient features. For example, the `Hubbard` matrix includes the parameter `ranpot` to switch on a random potential. Random numbers in ScaMaC are entirely reproducible, and independent of the number of threads or processes that call the ScaMaC routines, or of the order in which the

matrix rows are generated. An identical random seed gives the same matrix under all circumstances. In particular, individual matrix rows can be reconstructed at any time, which simplifies a fault-tolerant program design (see Sect. 5.3). Another feature is the possibility to effortlessly generate the (conjugate) transpose of non-symmetric (non-Hermitian) matrices, which is considerably easier than constructing the transpose of a (distributed) sparse matrix after generation.

6 Application Results

6.1 Eigensolvers in Quantum Physics: Graphene, Topological Insulators, and Beyond

Because of the linearity of the Schrödinger equation, quantum physics is a paradigm for numerical linear algebra applications. Historically, some application cases, such as the computation of the ground state (i.e. of the eigenvector to the minimal eigenvalue), have received so much attention that only gradual progress remains possible nowadays. In the ESSEX project we instead address two major cases where novel algorithmic improvements and systematic utilization of large-scale computing resources through state-of-the-art implementations still result in substantial qualitative progress. These two cases are the computation of (i) extreme eigenvalues with high degeneracy, which is addressed with a block Jacobi–Davidson algorithm, (ii) multiple interior eigenvalues, which is addressed by various filter diagonalization techniques. Application case (i) has been documented in [64], including the example of spin chain matrices (`SpinChainXXZ` in the ScaMaC). For application case (ii) the primary quantum physics example are graphene [13] and topological insulators [31] (`Graphene` and `TopIns` in the ScaMaC). For these examples, eigenvalues towards the center of the spectrum, near the Fermi energy of the material, are those of interest. This situation is similar to applications in quantum chemistry and density functional theory, but in our case the matrices represent a full (disordered or structured) two or three-dimensional domain, and are usually larger than those considered elsewhere [57].

Starting with the paper [62] on Chebyshev filter diagonalization (ChebFD) and culminating in the BEAST software package (see Sect. 3.2), the computation of interior eigenvalues of large-to-huge graphene and topological insulator matrices has been successfully demonstrated with ESSEX algorithms, using polynomial filters derived from Chebyshev polynomials. Already with the simple ChebFD algorithm we could compute $N_T \simeq 100$ eigenvectors from the center of the spectrum of a matrix with dimension $D \simeq 10^9$ (i.e. an effective problem size $N_T \times D \simeq 10^{11}$), in order to understand the electronic properties of a structured topological insulator (see Figure 13 in [62]). With improved filter coefficients and a more sophisticated implementation, the polynomial filters in the (P-) BEAST package deal with such problems at reduced computational cost (see Sect. 3.2).

www.dbooks.org

Such large-scale computations heavily rely on the optimized spMMVM and TSMM kernels of the GHOST library (see Sect. 5).

To appreciate the numerical progress reflected in these numbers one should note the different scaling of the numerical effort N_{MVM} (measured in terms of the dominant operation of spMVM) for the computation of extreme and interior eigenvalues (cf. the discussion in [62]). In an idealized situation with equidistant eigenvalues, we have roughly $N_{\mathrm{MVM}} \sim D^{1/2}$ for extreme but $N_{\mathrm{MVM}} \sim D$ for interior eigenvalues. For the $D \simeq 10^9$ example, we have to compensate for a factor 10^4–10^5 to enable computation of interior instead of extreme eigenvalues.

Algorithm and software development in ESSEX has been to a large degree application-driven. Now, at the end of the ESSEX project, where the algorithms for our main application cases have become available, we follow two ways to go beyond the initial quantum physics applications. First, entirely new applications can now be addressed with ESSEX software, extending our efforts to non-linear and non-Hermitian problems (see Sect. 6.2). Second, relevant applications such as the `Hubbard` and `Exciton` examples (see Sect. 5.5) still fit into the two major application areas already addressed in ESSEX, but further increase the computational complexity. For `Exciton`, the strongly clustered spectrum with many nearly-degenerate eigenvalues leads to a numerical effort $N_{\mathrm{MVM}} \gg D^{1/2}$ already for extreme eigenvalues. For `Hubbard`, the huge matrix dimension D is a serious obstacle for the computation of interior eigenvalues.

The `Hubbard` matrices also hint at an application-specific issue of general interest that we encountered but could not solve within ESSEX. Specifically, it is the complicated sparsity pattern of many of our quantum physics matrices (see Fig. 19) that adversely affects the parallel efficiency of distributed spMVM, and thus of our entire software solutions. Node-level performance engineering is here easily overcompensated by communication overhead. Unfortunately, the communication overhead is not reduced by standard matrix reordering strategies [61, 71, 84]. This problem can be partially alleviated by overlapping communication with computation, as in the spMMVM (see Sect. 2), but a full solution to restore parallel efficiency is not yet available. Clearly, our different application scenarios still provide enough incentive to think about future numerical, algorithmic, and computational developments beyond the ESSEX project.

6.2 New Applications in Nonlinear Dynamical Systems

The block Jacobi–Davidson QR eigensolver in PHIST is capable of solving non-symmetric and generalized eigenvalue problems of the form

$$\mathbf{A}\mathrm{x} = \lambda \mathbf{B}\mathrm{x}, \tag{1}$$

where $\mathbf{B}$ should be symmetric and positive definite. In [75], we exploited several unique features of this implementation to study the linear stability of a

three-dimensional reaction-diffusion equation: the Jacobian is non-symmetric, the preconditioner was implemented in Epetra (which can be used directly as a backend for PHIST), and the high degree of symmetry in the model yields eigenvalues with high geometric multiplicity (up to 24). We therefore use a relatively large block size of 8 for these computations to achieve convergence to the desired 20–50 eigenpairs $(\lambda_i, \mathrm{x}_i)$, with the real part of λ_i near 0. In a recent Ph.D. thesis [74], the solver was also used for studying the linear stability of incompressible flow problems. Here **B** is in fact only semi-definite, and the preconditioner has to make sure that the solution stays in the 'divergence-free space', in which the velocity field satisfies $\nabla \cdot \mathrm{u} = 0$ and **B** induces a norm.

Another ongoing effort concerning dynamical systems is the use of PHIST to parallelize the dynamical systems analysis tool PyNCT, which has as its main application the study of superconductors [82]. We have taken first steps to use PHIST as backend for the Python-based algorithms in PyNCT. Furthermore, it is possible to solve the eigenvalue problems arising in PyNCT directly by the BJDQR method in PHIST. Our goal here is the scalable parallel and fully automatic computation of bifurcation diagrams using PyNCT and any backend supported by PHIST.

The Statistical Learning Lab led by Dr. Marina Meila at the University of Washington started to use the PHIST eigensolver to compute spectral gaps for Laplacian matrices obtained from conformation trajectories in molecular dynamics simulations, and other scientific data [14, 59]. These are symmetric positive definite matrices whose dimensions equal the number of simulation steps, typically of the order of $n = 10^6$. When the data intrinsic dimension d is fixed, and much smaller than n (in our examples $d < 10$), the Laplacian is a sparse matrix. The sparsity pattern is not regular, and it is data dependent, as it reflects the neighborhood relationships in the data. Hence, in densely sampled regions rows will have many more non-zeros than in the sparsely populated regions of the data. In a manifold embedding algorithm, the eigengaps identify the optimal number of coordinates in which to embed the data. Furthermore, for data sizes $n \gg 10^6$, PHIST is used to compute the diffusion map embedding itself for the higher frequency coordinates for which existing methods are prohibitively slow.

7 International Collaborations

The internationalisation effort in the second phase of SPPEXA has fostered the ESSEX-II activities in several directions. First and foremost it amplified the scientific expertise in the project. Soon it became clear that complementing knowledge and developments could be leveraged across the partners. A specific benefit of the collaboration between German and Japanese partners is their very different background in terms of HPC infrastructures.Through close personal collaboration within the project all partners could easily access and use latest supercomputers on either side (see Sect. 4.2 and note that the BEAST framework has also been ported

www.dbooks.org

to the K-computer). Together with the joint collaboration on scientific problems and software development a steady exchange evolved with many personal research visits which also opened up collaboration with partners not involved directly in ESSEX-II.

The results described in Sect. 3.1 on preconditioners are a direct result of the collaboration of ESSEX-II with the ppOpen-HPC[18] project led by Univ. of Tokyo. On the other hand, the CRAFT library developed at Univ. of Erlangen is utilized in an FEM code of Univ. of Tokyo and is part of a follow on JHPCN project with the German partner involved as associated partner.

Collaboration between Japanese and German working groups made possible the expansion of the BEAST framework for projection based eigensolvers to include Sakurai–Sugiura methods. Various numerical and theoretical issues associated with the implementation of the solver within an iterative framework were resolved, and new ideas explored during research visits. Results based on this collaboration have so far been presented in multiple conferences and a paper in preparation [35].

The linear systems arising from numerical quadrature in the BEAST-C and BEAST-M framework were used in the testing and development of a Block Cholesky-based ILU preconditioner. The integration of an interface to this solver into BEAST has begun. Examining strategies and expectations for solving these extreme ill-conditioned problems was a point of intense discussion and collaboration between working groups. One results was the development of RACE (see Sect. 3.4). Beyond the discussion between several Japanese and German ESSEX-II partners also a strong collaboration with the Swiss partner (O. Schenk) of EXASTEEL-II evolved, who is an expert on direct solvers and graph partitioning. In this context also a collaboration with T. Iwashita (Hokkaido Univ., Japan) started in terms of hardware efficient coloring.

Throughout the project, the variety of large matrices continuously added to the ScaMaC library allowed for testing with a variety of realistically challenging problems of both real and complex types in all ESSEX-II working groups.

Acknowledgments The project is funded by the German DFG priority programme 1648 (Software for Exascale Computing) under the ESSEX-I/II (Equipping Sparse Solvers for Exascale) projects. We are grateful for computer time granted on the LRZ SuperMUC and SuperMUC-NG, the CSCS Piz Daint, and the OakForest PACS systems.

References

1. Alappat, C.L., Hager, G., Schenk, O., Thies, J., Basermann, A., Bishop, A.R., Fehske, H., Wellein, G.: A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. Accepted for publication in ACM Transactions on Parallel Computing. Preprint: http://arxiv.org/abs/1907.06487

[18] http://ppopenhpc.cc.u-tokyo.ac.jp/ppopenhpc/.

2. Alvermann, A., Fehske, H.: Exciton mass and exciton spectrum in the cuprous oxide. J. Phys. B **51**(4), 044001 (2018). https://doi.org/10.1088/1361-6455/aaa060. http://stacks.iop.org/0953-4075/51/i=4/a=044001
3. Alvermann, A., Basermann, A., Fehske, H., Galgon, M., Hager, G., Kreutzer, M., Krämer, L., Lang, B., Pieper, A., Röhrig-Zöllner, M., Shahzad, F., Thies, J., Wellein, G.: ESSEX: equipping sparse solvers for exascale. In: Lopes, L. et al. (eds.) Euro-Par 2014: Parallel Processing Workshops. Lecture Notes in Computer Science, vol. 8806, pp. 577–588. Springer, Berlin (2014). http://doi.org/10.1007/978-3-319-14313-2_49
4. Alvermann, A., Basermann, A., Bungartz, H.J., Carbogno, C., Ernst, D., Fehske, H., Futamura, Y., Galgon, M., Hager, G., Huber, S., Huckle, T., Ida, A., Imakura, A., Kawai, M., Köcher, S., Kreutzer, M., Kus, P., Lang, B., Lederer, H., Manin, V., Marek, A., Nakajima, K., Nemec, L., Reuter, K., Rippl, M., Röhrig-Zöllner, M., Sakurai, T., Scheffler, M., Scheurer, C., Shahzad, F., Simoes Brambila, D., Thies, J., Wellein, G.: Benefits from using mixed precision computations in the ELPA-AEO and ESSEX-II eigensolver projects. Jpn. J. Ind. Appl. Math. **36**, 699–717 (2019). https://doi.org/10.1007/s13160-019-00360-8
5. Anzt, H., Tomov, S., Dongarra, J.: Accelerating the LOBPCG method on GPUs using a blocked sparse matrix vector product. University of Tennessee Innovative Computing Laboratory Technical Report UT-EECS-14-731 (2014). http://www.eecs.utk.edu/resources/library/589
6. Anzt, H., Tomov, S., Dongarra, J.: Implementing a sparse matrix vector product for the SELL-C/SELL-C-σ formats on NVIDIA GPUs. University of Tennessee Innovative Computing Laboratory Technical Report UT-EECS-14-727 (2014). http://www.eecs.utk.edu/resources/library/585
7. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for nonlinear eigenvalue problems using contour integrals. JSIAM Lett. **1**, 52–55 (2009). https://doi.org/10.14495/jsiaml.1.52
8. Asakura, J., Sakurai, T., Tadano, H., Ikegami, T., Kimura, K.: A numerical method for polynomial eigenvalue problems using contour integral. Jpn. J. Ind. Appl. Math. **27**(1), 73–90 (2010). https://doi.org/10.1007/s13160-010-0005-x
9. Balay, S., Abhyankar, S., Adams, M.F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W.D., Karpeyev, D., Kaushik, D., Knepley, M.G., May, D.A., McInnes, L.C., Mills, R.T., Munson, T., Rupp, K., Sanan, P., Smith, B.F., Zampini, S., Zhang, H., Zhang, H.: PETSc Web page (2019). https://www.mcs.anl.gov/petsc
10. Barel, M.V., Kravanja, P.: Nonlinear eigenvalue problems and contour integrals. J. Comput. Appl. Math. **292**, 526–540 (2016). https://doi.org/10.1016/j.cam.2015.07.012
11. Bartlett, R., Demeshko, I., Gamblin, T., Hammond, G., Heroux, M., Johnson, J., Klinvex, A., Li, X., McInnes, L., Moulton, J.D., Osei-Kuffuor, D., Sarich, J., Smith, B., Willenbring, J., Yang, U.M.: xSDK foundations: toward an extreme-scale scientific software development kit. Supercomput. Front. Innov. Int. J. **4**(1), 69–82 (2017). https://doi.org/10.14529/jsfi170104
12. Beyn, W.J.: An integral method for solving nonlinear eigenvalue problems. Linear Algebra Appl. **436**(10), 3839–3863 (2012). https://doi.org/10.1016/j.laa.2011.03.030. Special Issue dedicated to Heinrich Voss's 65th birthday
13. Castro Neto, A.H., Guinea, F., Peres, N.M.R., Novoselov, K.S., Geim, A.K.: The electronic properties of graphene. Rev. Mod. Phys. **81**, 109–162 (2009). https://doi.org/10.1103/RevModPhys.81.109
14. Chen, Y.C., Meilă, M.: Selecting the independent coordinates of manifolds with large aspect ratios (2019, e-prints). arXiv:1907.01651.
15. Chen, H., Imakura, A., Sakurai, T.: Improving backward stability of Sakurai-Sugiura method with balancing technique in polynomial eigenvalue problem. Appl. Math. **62**(4), 357–375 (2017). https://doi.org/10.21136/AM.2017.0016-17
16. Chen, H., Maeda, Y., Imakura, A., Sakurai, T., Tisseur, F.: Improving the numerical stability of the Sakurai-Sugiura method for quadratic eigenvalue problems. JSIAM Lett. **9**, 17–20 (2017). https://doi.org/10.14495/jsiaml.9.17
17. Davis, T.A., Hu, Y.: The University of Florida sparse matrix collection. ACM Trans. Math. Softw. **38**(1), 1:1–1:25 (2011). https://doi.org/10.1145/2049662.2049663

www.dbooks.org

18. Druskin, V., Knizhnerman, L.: Two polynomial methods to compute functions of symmetric matrices. U.S.S.R. Comput. Maths. Math. Phys. **29**(6), 112–121 (1989). https://doi.org/10.1016/S0041-5553(89)80020-5
19. Edwards, H.C., Trott, C.R., Sunderland, D.: Kokkos: enabling manycore performance portability through polymorphic memory access patterns. J. Parallel Distrib. Comput. **74**(12), 3202–3216 (2014). https://doi.org/10.1016/j.jpdc.2014.07.003. Domain-Specific Languages and High-Level Frameworks for High-Performance Computing
20. Ernst, D., Hager, G., Thies, J., Wellein, G.: Performance engineering for a tall & skinny matrix multiplication kernel on GPUs. In: Proceedings PPAM'19: the 13h International Conference on Parallel Processing and Applied Mathematics, Bialystok, Poland, September 8–11, 2019. https://doi.org/10.1007/978-3-030-43229-4_43
21. EXASTEEL project website: www.numerik.uni-koeln.de/14426.html
22. Fukasawa, T., Shahzad, F., Nakajima, K., Wellein, G.: pFEM-CRAFT: a library for application-level fault-resilience based on the CRAFT framework. In: Poster at the 2018 SIAM Conference on Parallel Processing for Scientific Computing (SIAM PP18), Tokyo (2018)
23. Galgon, M., Krämer, L., Lang, B., Alvermann, A., Fehske, H., Pieper, A.: Improving robustness of the FEAST algorithm and solving eigenvalue problems from graphene nanoribbons. PAMM **14**(1), 821–822 (2014). http://doi.org/10.1002/pamm.201410391
24. Galgon, M., Krämer, L., Thies, J., Basermann, A., Lang, B.: On the parallel iterative solution of linear systems arising in the FEAST algorithm for computing inner eigenvalues. Parallel Comput. **49**, 153–163 (2015)
25. Galgon, M., Krämer, L., Lang, B., Alvermann, A., Fehske, H., Pieper, A., Hager, G., Kreutzer, M., Shahzad, F., Wellein, G., Basermann, A., Röhrig-Zöllner, M., Thies, J.: Improved coefficients for polynomial filtering in ESSEX. In: Sakurai, T., Zhang, S.L., Imamura, T., Yamamoto, Y., Kuramashi, Y., Hoshi, T. (eds.) Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, pp. 63–79. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62426-6_5
26. Galgon, M., Krämer, L., Lang, B.: Improving projection-based eigensolvers via adaptive techniques. Numer. Linear Algebra Appl. **25**(1), e2124 (2018). http://dx.doi.org/10.1002/nla.2124
27. Gamblin, T., LeGendre, M.P., Collette, M.R., Lee, G.L., Moody, A., de Supinski, B.R., Futral, W.S.: The Spack package manager: bringing order to HPC software chaos (2015). LLNL-CONF-669890
28. Giorgi, P., Vialla, B.: Generating optimized sparse matrix vector product over finite fields. In: Proceedings of ICMS 2014: Fourth International Congress on Mathematical Software, Seoul. Lecture Notes in Computer Science, vol. 8592, pp. 685–690. Springer, Berlin (2014). http://www.lirmm.fr/~giorgi/icms2014-giovia.pdf
29. Gordon, D., Gordon, R.: CGMN revisited: robust and efficient solution of stiff linear systems derived from elliptic partial differential equations. ACM Trans. Math. Softw. **35**(3), 18:1–18:27 (2008). https://doi.org/10.1145/1391989.1391991
30. Guettel, S., Polizzi, E., Tang, P., Viaud, G.: Zolotarev quadrature rules and load balancing for the FEAST eigensolver. SIAM J. Sci. Comput. **37**(4), A2100–A2122 (2015). https://doi.org/10.1137/140980090
31. Hasan, M.Z., Kane, C.L.: Colloquium: topological insulators. Rev. Mod. Phys. **82**, 3045–3067 (2010). https://doi.org/10.1103/RevModPhys.82.3045
32. Hasegawa, T., Imakura, A., Sakurai, T.: Recovering from accuracy deterioration in the contour integral-based eigensolver. JSIAM Lett. **8**, 1–4 (2016). https://doi.org/10.14495/jsiaml.8.1
33. Heroux, M.A., Bartlett, R.A., Howle, V.E., Hoekstra, R.J., Hu, J.J., Kolda, T.G., Lehoucq, R.B., Long, K.R., Pawlowski, R.P., Phipps, E.T., Salinger, A.G., Thornquist, H.K., Tuminaro, R.S., Willenbring, J.M., Williams, A., Stanley, K.S.: An overview of the Trilinos project. ACM Trans. Math. Softw. **31**(3), 397–423 (2005). http://doi.acm.org/10.1145/1089014.1089021
34. Hubbard, J., Flowers, B.H.: Electron correlations in narrow energy bands. Proc. Roy. Soc. Lond. A **276**(1365), 238–257 (1963). https://doi.org/10.1098/rspa.1963.0204

35. Huber, S., Futamura, Y., Galgon, M., Imakura, A., Lang, B., Sakurai, T.: Flexible subspace iteration with moments for an effective contour-integration based eigensolver (2019, in preparation)
36. Ikegami, T., Sakurai, T.: Contour integral eigensolver for non-hermitian systems: a Rayleigh-Ritz-type approach. Taiwan. J. Math. **14**(3A), 825–837 (2010). http://www.jstor.org/stable/43834819
37. Ikegami, T., Sakurai, T., Nagashima, U.: A filter diagonalization for generalized eigenvalue problems based on the Sakurai–Sugiura projection method. J. Comput. Appl. Math. **233**(8), 1927–1936 (2010). https://doi.org/10.1016/j.cam.2009.09.029
38. Imakura, A., Sakurai, T.: Block Krylov-type complex moment-based eigensolvers for solving generalized eigenvalue problems. Numer. Algorithms **75**(2), 413–433 (2017). https://doi.org/10.1007/s11075-016-0241-5
39. Imakura, A., Sakurai, T.: Block SS–CAA: a complex moment-based parallel nonlinear eigensolver using the block communication-avoiding Arnoldi procedure. Parallel Comput. **74**, 34–48 (2018). https://doi.org/10.1016/j.parco.2017.11.007. Parallel Matrix Algorithms and Applications (PMAA'16)
40. Imakura, A., Du, L., Sakurai, T.: A block Arnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems. Appl. Math. Lett. **32**, 22–27 (2014). https://doi.org/10.1016/j.aml.2014.02.007
41. Imakura, A., Du, L., Sakurai, T.: Error bounds of Rayleigh–Ritz type contour integral-based eigensolver for solving generalized eigenvalue problems. Numer. Algorithms **71**(1), 103–120 (2016). https://doi.org/10.1007/s11075-015-9987-4
42. Imakura, A., Du, L., Sakurai, T.: Relationships among contour integral-based methods for solving generalized eigenvalue problems. Jpn. J. Ind. Appl. Math. **33**(3), 721–750 (2016). https://doi.org/10.1007/s13160-016-0224-x
43. Imakura, A., Futamura, Y., Sakurai, T.: An error resilience strategy of a complex moment-based eigensolver. In: Sakurai, T., Zhang, S.L., Imamura, T., Yamamoto, Y., Kuramashi, Y., Hoshi, T. (eds.) Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing, pp. 1–18. Springer, Cham (2017)
44. Iwashita, T., Nakashima, H., Takahashi, Y.: Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in ICCG method. In: Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS'12, pp. 474–483. IEEE Computer Society, Washington (2012). https://doi.org/10.1109/IPDPS.2012.51
45. Kawai, M., Iwashita, T., Nakashima, H., Marques, O.: Parallel smoother based on block red-black ordering for multigrid Poisson solver. In: High Performance Computing for Computational Science – VECPAR 2012, pp. 292–299 (2013)
46. Kawai, M., Ida, A., Nakajima, K.: Hierarchical parallelization of multi-coloring algorithms for block IC preconditioners. In: 2017 IEEE 19th International Conference on High Performance Computing and Communications; IEEE 15th International Conference on Smart City; IEEE 3rd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 138–145. IEEE, Piscataway (2017). https://doi.org/10.1109/HPCC-SmartCity-DSS.2017.18
47. Kawai, M., Ida, A., Nakajima, K.: Modified IC preconditioner of CG method for ill-conditioned problems (in Japanese). Tech. Rep. vol. 2017-HPC-158, No.9, IPSJ SIG (2017)
48. Kawai, M., Ida, A., Nakajima, K.: Higher precision for block ILU preconditioner. In: CoSaS2018. FAU (2018)
49. Krämer, L.: Integration based solvers for standard and generalized Hermitian eigenvalue problems. Ph.D. Thesis, Bergische Universität Wuppertal (2014). http://nbn-resolving.de/urn/resolver.pl?urn=urn:nbn:de:hbz:468-20140701-112141-6
50. Krämer, L., Di Napoli, E., Galgon, M., Lang, B., Bientinesi, P.: Dissecting the FEAST algorithm for generalized eigenproblems. J. Comput. Appl. Math. **244**, 1–9 (2013)
51. Kreutzer, M.: Performance engineering for exascale-enabled sparse linear algebra building blocks. Ph.D. Thesis, FAU Erlangen-Nürnberg, Technische Fakultät, Erlangen (2018). https://doi.org/10.25593/978-3-96147-104-1

www.dbooks.org

52. Kreutzer, M., Hager, G., Wellein, G., Fehske, H., Bishop, A.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units. SIAM J. Sci. Comput. **36**(5), C401–C423 (2014). https://doi.org/10.1137/130930352
53. Kreutzer, M., Pieper, A., Hager, G., Wellein, G., Alvermann, A., Fehske, H.: Performance engineering of the Kernel Polynomal Method on large-scale CPU-GPU systems. In: 2015 IEEE International Parallel and Distributed Processing Symposium, pp. 417–426 (2015). https://doi.org/10.1109/IPDPS.2015.76
54. Kreutzer, M., Thies, J., Pieper, A., Alvermann, A., Galgon, M., Röhrig-Zöllner, M., Shahzad, F., Basermann, A., Bishop, A.R., Fehske, H., Hager, G., Lang, B., Wellein, G.: Performance engineering and energy efficiency of building blocks for large, sparse eigenvalue computations on heterogeneous supercomputers. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing—SPPEXA 2013–2015, pp. 317–338. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40528-5_14
55. Kreutzer, M., Thies, J., Röhrig-Zöllner, M., Pieper, A., Shahzad, F., Galgon, M., Basermann, A., Fehske, H., Hager, G., Wellein, G.: GHOST: building blocks for high performance sparse linear algebra on heterogeneous systems. Int. J. Parallel Program. **45**(5), 1046–1072 (2017). https://doi.org/10.1007/s10766-016-0464-z
56. Kreutzer, M., Ernst, D., Bishop, A.R., Fehske, H., Hager, G., Nakajima, K., Wellein, G.: Chebyshev filter diagonalization on modern manycore processors and GPGPUs. In: Yokota, R., Weiland, M., Keyes, D., Trinitis, C. (eds.) High Performance Computing, pp. 329–349. Springer, Cham (2018). https://dx.doi.org/10.1007/978-3-319-92040-5_17
57. Li, R., Xi, Y., Erlandson, L., Saad, Y.: The Eigenvalues Slicing Library (EVSL): algorithms, implementation, and software (preprint). http://www-users.cs.umn.edu/~saad/software/EVSL/index.html
58. Matrix Market: https://math.nist.gov/MatrixMarket/. Accessed 26 July 2019
59. Meila, M., Koelle, S., Zhang, H.: A regression approach for explaining manifold embedding coordinates (2018, e-prints). arXiv:1811.11891
60. Müthing, S., Ribbrock, D., Göddeke, D.: Integrating multi-threading and accelerators into DUNE-ISTL. In: Abdulle, A., Deparis, S., Kressner, D., Nobile, F., Picasso, M., Quarteroni, A. (eds.) Numerical Mathematics and Advanced Applications – ENUMATH 2013. Lecture Notes in Computational Science and Engineering, vol. 103, pp. 601–609. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-10705-9_59
61. ParMETIS - parallel graph partitioning and fill-reducing matrix ordering. http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview
62. Pieper, A., Kreutzer, M., Alvermann, A., Galgon, M., Fehske, H., Hager, G., Lang, B., Wellein, G.: High-performance implementation of Chebyshev filter diagonalization for interior eigenvalue computations. J. Comput. Phys. **325**, 226–243 (2016). http://dx.doi.org/10.1016/j.jcp.2016.08.027
63. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Phys. Rev. B **79**(11), 115112 (2009). https://doi.org/10.1103/PhysRevB.79.115112
64. Röhrig-Zöllner, M., Thies, J., Kreutzer, M., Alvermann, A., Pieper, A., Basermann, A., Hager, G., Wellein, G., Fehske, H.: Increasing the performance of the Jacobi-Davidson method by blocking. SIAM J. Sci. Comput. **37**(6), 206–239 (2015). http://dx.doi.org/10.1137/140976017
65. Sakurai, T., Sugiura, H.: A projection method for generalized eigenvalue problems using numerical integration. J. Comput. Appl. Math. **159**(1), 119–128 (2003). https://doi.org/10.1016/S0377-0427(03)00565-X. Sixth Japan-China Joint Seminar on Numerical Mathematics; In Search for the Frontier of Computational and Applied Mathematics toward the 21st Century

66. Sakurai, T., Asakura, J., Tadano, H., Ikegami, T.: Error analysis for a matrix pencil of Hankel matrices with perturbed complex moments. JSIAM Lett. **1**, 76–79 (2009). https://doi.org/10.14495/jsiaml.1.76
67. Sakurai, T., Futamura, Y., Imakura, A., Imamura, T.: Scalable Eigen-Analysis Engine for Large-Scale Eigenvalue Problems, pp. 37–57. Springer, Singapore (2019). https://doi.org/10.1007/978-981-13-1924-2_3
68. Sato, K., et al.: Design and modeling of a non-blocking checkpointing system. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 19:1–19:10. IEEE Computer Society Press, Los Alamitos (2012)
69. Schenk, O., Gärtner, K., Fichtner, W.: Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors. BIT Numer. Math. **40**(1), 158–176 (2000). https://doi.org/10.1023/A:1022326604210
70. Schollwöck, U.: The density-matrix renormalization group. Rev. Mod. Phys. **77**, 259–315 (2005)
71. SCOTCH: Static mapping, graph, mesh and hypergraph partitioning, and parallel and sequential sparse matrix ordering package. http://www.labri.fr/perso/pelegrin/scotch/
72. Shahzad, F.: Checkpoint/restart and automatic fault tolerance (CRAFT) library. https://bitbucket.org/essex/craft. Accessed 27 July 2017
73. Shahzad, F., Thies, J., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: CRAFT: a library for easier application-level checkpoint/restart and automatic fault tolerance. IEEE Trans. Parallel Distrib. Syst. **30**(3), 501–514 (2019). https://doi.org/10.1109/TPDS.2018.2866794
74. Song, W.: Matrix-based techniques for (flow-)transition studies. Ph.D. Thesis, University of Groningen (2019). https://elib.dlr.de/125176/
75. Song, W., Wubs, F.W., Thies, J., Baars, S.: Numerical bifurcation analysis of a 3D Turing-type reaction-diffusion model. Commun. Nonlinear Sci. Numer. Simul. **60**, 145-164 (2018). https://doi.org/10.1016/j.cnsns.2018.01.003
76. Tang, P.T.P., Polizzi, E.: FEAST as a subspace iteration eigensolver accelerated by approximate spectral projection. SIAM J. Matrix Anal. Appl. **35**(2), 354–390 (2014). https://doi.org/10.1137/13090866X
77. Thies, J., Galgon, M., Shahzad, F., Alvermann, A., Kreutzer, M., Pieper, A., Röhrig-Zöllner, M., Basermann, A., Fehske, H., Hager, G., Lang, B., Wellein, G.: Towards an exascale enabled sparse solver repository (2016). In: Software for Exascale Computing – SPPEXA 2013-2015, Volume 113 of the series Lecture Notes in Computational Science and Engineering, 295-316 (2016). http://doi.org/10.1007/978-3-319-40528-5_13. Preprint: https://elib.dlr.de/100211/
78. Thies, J., Röhrig-Zöllner, M., Overmars, N., Basermann, A., Ernst, D., Hager, G., Wellein, G.: PHIST: a pipelined, hybrid-parallel iterative solver toolkit. Accepted for publication in ACM Trans. Math. Softw. Preprint: https://elib.dlr.de/123323/
79. Van der Vorst, H.A.: Iterative Krylov methods for large linear systems, vol. 13. Cambridge University Press, Cambridge (2003)
80. ViennaCL - the Vienna computing library. http://viennacl.sourceforge.net/doc/index.html
81. Weiße, A., Wellein, G., Alvermann, A., Fehske, H.: The kernel polynomial method. Rev. Mod. Phys. **78**, 275–306 (2006). https://link.aps.org/doi/10.1103/RevModPhys.78.275
82. Wouters, M., Vanroose, W.: Automatic exploration techniques for the numerical bifurcation study of the Ginzburg-Landau equation. SIAM J. Dynam. Syst. (2019, submitted). Preprint: https://arxiv.org/abs/1903.02377
83. Yokota, S., Sakurai, T.: A projection method for nonlinear eigenvalue problems using contour integrals. JSIAM Lett. **5**, 41–44 (2013). https://doi.org/10.14495/jsiaml.5.41
84. Zoltan: Parallel partitioning, load balancing and data-management services. http://www.cs.sandia.gov/zoltan/Zoltan.html

www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



ExaDG: High-Order Discontinuous Galerkin for the Exa-Scale



Daniel Arndt, Niklas Fehn, Guido Kanschat, Katharina Kormann, Martin Kronbichler, Peter Munch, Wolfgang A. Wall, and Julius Witte

Abstract This text presents contributions to efficient high-order finite element solvers in the context of the project ExaDG, part of the DFG priority program 1648 *Software for Exascale Computing* (SPPEXA). The main algorithmic components are the matrix-free evaluation of finite element and discontinuous Galerkin operators with sum factorization to reach a high node-level performance and parallel scalability, a massively parallel multigrid framework, and efficient multigrid smoothers. The algorithms have been applied in a computational fluid dynamics context. The software contributions of the project have led to a speedup by a factor $3-4$ depending on the hardware. Our implementations are available via the deal.II finite element library.

D. Arndt
Oak Ridge National Laboratory, Oak Ridge, TN, USA
e-mail: arndtd@ornl.gov

N. Fehn · M. Kronbichler (✉) · W. A. Wall
Technical University of Munich, Garching, Germany
e-mail: fehn@lnm.mw.tum.de; kronbichler@lnm.mw.tum.de; wall@lnm.mw.tum.de

K. Kormann
Max Planck Institute for Plasma Physics, Garching, Germany

Technical University Munich, Garching, Germany
e-mail: katharina.kormann@ipp.mpg.de; katharina.kormann@tum.de

P. Munch
Technical University of Munich, Garching, Germany

Helmholtz-Zentrum Geesthacht, Geesthacht, Germany
e-mail: munch@lnm.mw.tum.de; peter.muench@hzg.de

G. Kanschat · J. Witte
Heidelberg University, Heidelberg, Germany
e-mail: kanschat@uni-heidelberg.de; julius.witte@iwr.uni-heidelberg.de

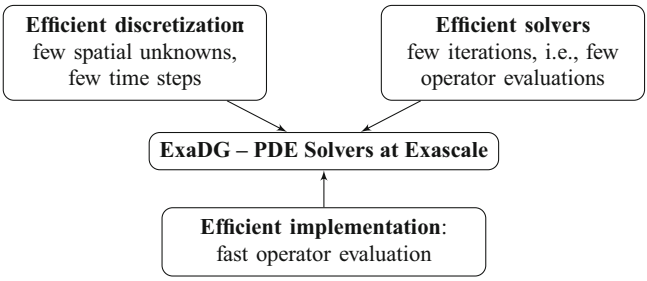
© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_8

www.dbooks.org

1 Introduction

Exa-scale performance of numerical algorithms is determined by two factors, node-level performance and distributed-memory scalability to thousands of nodes over an Infiniband-type fabric. Additionally, the final application efficiency in terms of time-to-solution is strongly influenced by the choice of numerical methods, where a high sequential efficiency is essential. The project ExaDG aims to bring together these three pillars to create an algorithmic framework for the next generation of solvers for partial differential equations (PDEs). The guiding principles of the project are as follows:



If we define the overall goal to be a minimum of computational cost to reach a predefined accuracy, this aim can be split into three components, namely the efficiency of the discretization in terms of the number of degrees of freedom (DoFs) and time steps, the efficiency of the solvers in terms of iteration counts, and the efficiency of the implementation [22]:

$$E = \frac{\text{accuracy}}{\text{computational cost}} = \underbrace{\frac{\text{accuracy}}{\text{DoFs} \cdot \text{timesteps}}}_{\text{discretization}} \cdot \underbrace{\frac{1}{\text{iterations}}}_{\text{solvers/preconditioners}} \cdot \underbrace{\frac{\text{DoFs} \cdot \text{timesteps} \cdot \text{iterations}}{\text{computational cost}}}_{\text{implementation}} . \tag{1}$$

We define computational cost as the product of compute resources (cores, nodes) times the wall time resulting in the metric of CPUh, the typical currency of supercomputing facilities.

Regarding the first metric, the type of discretizations in space and time are often the first decision to be made. Numerical schemes that involve as few unknowns and as few time steps as possible to reach the desired accuracy will be more efficient. This goal can be reached by using higher order methods which have a higher resolution capability, especially for problems with a large range of scales and some regularity in the solution [19]. However several possibilities and profound knowledge regarding the performance capability of potential algorithms on modern

hardware are still required to select those algorithms and implementations that are optimal with respect to the guiding metric of accuracy versus time-to-solution. High-order (dis-)continuous finite element methods are the basic building block of the ExaDG project due to their generality and geometric flexibility.

Regarding the second metric, solvers are deemed efficient if they keep the number of iterations minimal. We emphasize that "iterations" are defined in a low-level way as the number of operator evaluations, which is also accurate when nesting several iterative schemes within each other. Note that we assume that large-scale systems must be addressed by iterative solvers; in a finite element context sparse direct solvers are not scalable due to fill-in and complex dependencies during factorizations. One class of efficient solvers of particular interest to ExaDG are multigrid methods with suitable smoothers, which have developed to be the gold standard of solvers for elliptic and parabolic differential equations over the last decades. Here, the concept of iterations would accumulate several matrix-vector products within a multigrid cycle that in turn is applied in an outer Krylov subspace solver. Due to the grid transfer and the coarse grid solver, such methods are inherently challenging for highly parallel environments. As part of our efforts in ExaDG, we have developed an efficient yet flexible implementation in the deal.II finite element library [1, 15].

Third, the evaluation of discretized operators and smoothers remains the key component determining computational efficiency of a PDE solver. The relevant metric in this context is the throughput measured as the number of degrees of freedom (unknowns) processed per second (DoFs/s). An important contribution of our efforts is to both tune the implementation of a specific algorithm, but more importantly to also adapt algorithms towards a higher throughput. This means that an algorithm is preferred if it increases the DoFs/s metric, even if it leads to lower arithmetic performance in GFlop/s or lower memory throughput in GB/s. Operator evaluation in PDE solvers only involves communication with the nearest neighbors in terms of a domain decomposition of the mesh, which makes the node-level performance the primary concern in this regard. Since iterative solvers only require the action of the matrix on a vector (and a preconditioner), they are amenable to matrix-free evaluation where the final matrix entries are neither computed nor stored globally in memory in some generic sparse matrix format (e.g., compressed row storage). While matrix-free methods were historically often considered because they lower main memory requirements and allow to fit larger problems in memory [8], their popularity is currently increasing because they need to move less memory: Sparse matrix-vector products are limited by the memory bandwidth on all major computing platforms, so a matrix-free alternative promises to deliver a (much) higher performance.

The outline of this article is as follows. We begin with an introduction of matrix-free algorithms and a presentation of node-level performance results in Sect. 2. In Sect. 3, we describe optimizations of the conjugate gradient method for efficient memory access and communication. Next, we detail our multigrid developments, focusing on performance numbers and the massively parallel setup in Sect. 4 and on the development of better smoothers in Sect. 5. Application results in the field

www.dbooks.org

of computational fluid dynamics are presented in Sect. 6, where the efficiency and parallel scalability of our discontinuous Galerkin incompressbile turbulent flow solver are shown. An extension of the kernels to up to 6D PDEs is briefly presented in Sect. 7. We conclude with an outlook in Sect. 8.

2 Node-Level Performance Through Matrix-Free Implementation

An intuitive example of a matrix-free implementation is a finite difference method implemented by its stencil rather than an assembled sparse matrix [33]. For finite element discretizations with sufficient structure of the underlying mesh and low-order shape functions, a small number of stencils allows to represent the operator of a large-scale problem [8]. Such methods are used in the German exascale project TerraNeo, utilizing the regular data structures in hierarchical hybrid grids and embedded into a highly scalable multigrid solver for Stokes systems [31, 32]. By suitable interpolations, the stencils can be extended from the affine coarse grid assumption to also treat smoothly deformed geometries and variable coefficients [7].

For higher-order methods, finite element discretizations lead to fat stencils, making the direct evaluation inefficient even when done through stencils. An alternative matrix-free scheme used in ExaDG is to not compute the explicit DoF coupling and instead turn to integrals underlying the finite element scheme. As an example, we consider the constant-coefficient Laplacian

$$-\nabla^2 u = f \quad \text{in } \Omega, \quad u = g \quad \text{on } \partial\Omega, \tag{2}$$

whose weak form in a finite-dimensional setting is

$$(\nabla\varphi_i, \nabla u_h)_{\Omega_h} = (\varphi_i, f)_{\Omega_h}, \tag{3}$$

where $u_h(x) = \sum_{j=1:n} \varphi_j(x) u_j$ is the finite element interpolant of the solution with n degrees of freedom, φ_i denotes the test functions with $i = 1, \ldots, n$, f is some right hand side, and Ω_h is the finite element representation of the computational domain Ω. The left-hand side of this equation represents a finite element operator, mapping a vector of coefficients $u = [u_i]_i$ to an output vector $v = [v_i]_i$ by evaluating the weak form for all test functions φ_i separately. A matrix-free implementation is obtained by evaluating the element-wise integrals

$$\begin{aligned}\left[(\nabla\varphi_i, \nabla u_h)_{\Omega_h}\right]_{i=1:n} &= \sum_K \int_{\hat{K}} \left(\mathcal{J}_K^{\mathsf{T}} \hat{\nabla}\varphi_i\right)^{\mathsf{T}} \left(\mathcal{J}_K^{\mathsf{T}} \sum_{j=1}^{n_{\text{dof,ele}}} \hat{\nabla}\varphi_j u_j^{(K)}\right) \det(\mathcal{J}_K)\, \mathrm{d}\hat{x} \\ &\approx \sum_K I_K^{\mathsf{T}} \left[\sum_{q=1}^{n_q} (\hat{\nabla}\varphi_{i_K}(\hat{x}_q))^{\mathsf{T}} \underbrace{\mathcal{J}_K^{-1}\mathcal{J}_K^{-\mathsf{T}} \det(\mathcal{J}_K) w_q}_{\text{physics at quadrature point}} \sum_{j=1}^{n_{\text{dof,ele}}} \hat{\nabla}\varphi_j(\hat{x}_q) u_j^{(K)}\right]_{i_K=1:n_{\text{dof,ele}}}\end{aligned} \tag{4}$$

by quadrature on n_q points per cell K. Here, K denotes the elements in the mesh, $\hat{x}$ the coordinates of the reference element $\hat{K} = (0,1)^d$, $\mathcal{J}_K$ denotes the Jacobian of the mapping from the reference to the real cell, and w_q the quadrature weight. The operator I_K denotes the index mapping from $n_{\text{dof,ele}}$ element-local to global unknowns and defines the element-related unknowns $u^{(K)} = I_K u$.

On element K, the formulation of Eq. (4) consists of two nested sums over the elemental unknowns $u_j^{(K)}$, $j \in n_{\text{dof,ele}}$, and the quadrature points q. The result is tested against all test functions φ_{i_K} on the reference element, which are related to the global test functions φ_i through I_K. Since the metric terms do not depend on the shape function indices i_K and j, and the sum over j does not depend on i_K, the summations in the equation can be broken up into (1) an $dn_q \times n_{\text{dof,ele}}$ matrix operation to evaluate the reference element derivative of $u^{(K)}$ at the quadrature points, (2) the application of metric terms as well as other physics terms at n_q quadrature points, and (3) an $n_{\text{dof,ele}} \times dn_q$ matrix operation to test by all $n_{\text{dof,ele}}$ test functions and perform the summation over the quadrature points. The separation of point-wise physics evaluation at quadrature points is a common abstraction in integration-based matrix-free methods [29, 41, 49, 50].

For high-order finite element methods, the naive evaluation would involve all shape functions at all quadrature points, which is of complexity $O(k^{2d})$ for polynomials of degree k in d dimensions per element, or $O(k^d)$ per unknown, similarly to the fat stencil of the final matrix.

At this point, the structure in the reference-cell shape function and quadrature points can be utilized to lower the computational complexity. If the multi-dimensional shape functions are the tensor product of 1D shape functions, and if the quadrature formula is a tensor product of 1D formulas, the so-called sum-factorization algorithm can be used to group common factors along the various dimensions and break down the work into one-dimensional interpolations. Figure 1 visualizes the process of computing the interpolation of nodal values, visualized by black disks, to the values at the quadrature points. Rather than using a naive interpolation of cost $2(k+1)^{2d}$ operations, it can be done in $2d(k+1)^{d+1}$ operations instead. In matrix-vector notation, the interpolation of the gradient with respect to

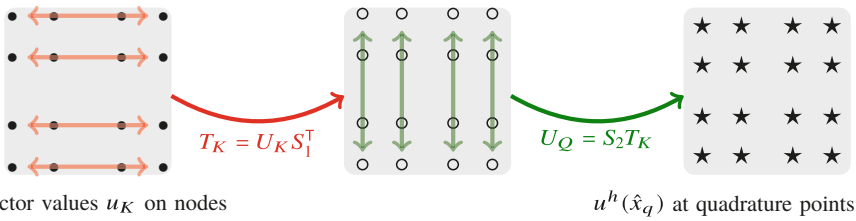


Fig. 1 Illustration of sum factorization for interpolation from node values on the left to the values in quadrature points (right)

www.dbooks.org

$\hat{x}$, evaluated at quadrature points, can be written as

$$\begin{bmatrix} \partial \boldsymbol{u}_h/\partial \hat{x}_1 \\ \partial \boldsymbol{u}_h/\partial \hat{x}_2 \\ \vdots \\ \partial \boldsymbol{u}_h/\partial \hat{x}_d \end{bmatrix} = \begin{bmatrix} I \otimes \ldots \otimes I \otimes D_1 \\ I \otimes \ldots \otimes D_2 \otimes I \\ \vdots \\ D_d \otimes I \otimes \ldots \otimes I \end{bmatrix} \left[S_d \otimes \ldots \otimes S_2 \otimes S_1 \right] \boldsymbol{u}^{(K)}. \tag{5}$$

Here, $S_1, \ldots, S_d$ denote the $n_q^{1\mathrm{D}} \times (k+1)$ interpolation matrices from the nodal values to the quadrature points, obtained by evaluating the 1D basis at all 1D quadrature points, and $D_1, \ldots, D_d$ the $n_q^{1\mathrm{D}} \times n_q^{1\mathrm{D}}$ matrices of the derivatives of the Lagrangian basis in quadrature points. In this form, the multiplication by Kronecker matrices is implemented by small matrix-matrix multiplications.

Sum factorization was initially developed in the context of spectral element methods by Orszag [61], see [19] for an overview of the developments. In [12], sum factorization was compared against assembled matrices with the goal to find the best evaluation strategy among assembled matrices and matrix-free schemes. For hexahedral elements considered in this work, the memory consumption and arithmetic complexity indicate that this is the case already for quadratic basis functions [11, 49], with a growing gap for higher polynomial degrees.

2.1 Implementation of Sum Factorization in the deal.II Library

As part of the ExaDG project, we have developed efficient implementations in the deal.II finite element library [1, 4] with the following main features, see [50] for a detailed performance analysis:

- support for both continuous [49] and discontinuous finite elements on uniform and adaptively refined meshes with hanging nodes and deformed elements,
- support for arbitrary polynomial expansions on quadrilateral and hexahedral element shapes as well as tensor product quadrature rules,
- minimization of arithmetic operations by using available symmetries, such as the even-odd decomposition [69] and a switch between the collocation derivative (5) for $n_q^{1\mathrm{D}} \approx k+1$ quadrature points or an alternative variant based on derivatives of the original polynomials as used in [49] and discussed in [29],
- flexible implementation of operations at quadrature points,
- vectorization across several elements to optimally use SIMD units (AVX, AVX-512, AltiVec) of modern processors,
- applicability to modern multi-core CPUs as well as GPUs [51, 57],
- data access optimizations such as element-based loops for DG elements [50, 56],
- and MPI implementation with tight data exchange as well as MPI-only and shared-memory models [43, 48, 54].

The concept of matrix-free evaluation with sum factorization has been widely adopted by now, like in the deal.II [1], DUNE [5, 40, 60], Firedrake [63], mfem [2], Nek5000 [28] or Nektar++ [13] projects. These fast evaluation techniques are directly applicable to explicit time stepping schemes, as we have demonstrated for wave propagation in [42, 53, 65–68] and the compressible Navier–Stokes equations [24]. The proposed developments make matrix-free evaluation of high-order DG operators reach a throughput in unknowns per second almost as high as for optimized 5-wide finite difference stencils in a CFD context [75], despite delivering much higher accuracy.

2.2 *Efficiency of Matrix-Free Implementation*

In Fig. 2, we give an overview of the achieved performance with our framework applied to the discontinuous Galerkin interior penalty (IPDG) discretization of the 3D Laplacian on an affine geometry. The most advanced implementation presented in [50] is used, namely a cell-based loop with a Hermite-like basis for minimal data access [56]. The figure lists the throughput, which is measured by recording the run time of the matrix-vector product in an experiment with around 50 million DoFs (too large to fit into caches), and reporting the normalized quantity DoFs/s obtained by dividing the number of DoFs by the measured run time. The code is run on a single node of six dual-socket HPC systems from the last decade with a shared-memory parallelization with OpenMP, threads pinned to logical cores with the close affinity rule, and using streaming stores to avoid the read-for-ownership data transfer [33] on the result vector. As systems, we consider a 2×8 core AMD Opteron 6128 system from 2010, a 2×8 core Intel Xeon 2680 Sandy Bridge from 2012 (as used in the SuperMUC phase 1 installation in Garching, Germany), a 2×8 core Intel Xeon 2630 v3 (Haswell) representing a medium-core count chip from

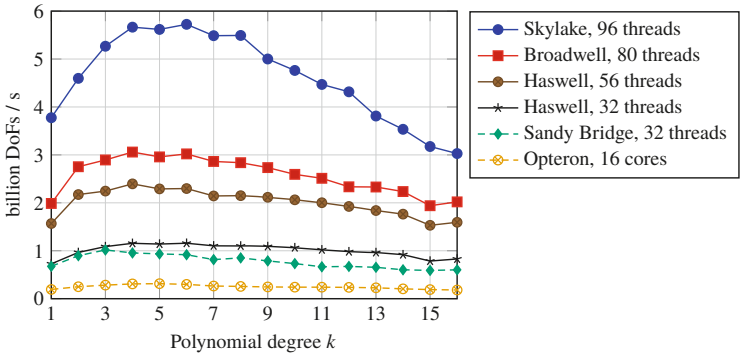


Fig. 2 Throughput of matrix-free evaluation of the IPDG discretization of the 3D Laplacian on an affine grid

www.dbooks.org

2014, a 2 × 14 core Intel Xeon 2697 v3 (Haswell) representing a high-core count chip of the same generation (as used in the SuperMUC phase 2 installation), a 2 × 20 core Intel Xeon 2698 v4 (Broadwell) system from 2016, and a 2 × 24 core Intel Xeon Platinum 8174 from 2017, labeled 'Skylake' in the remainder of this work, and installed in the SuperMUC-NG supercomputer. The chips are operated at 2.0 GHz, 2.7 GHz, 2.4 GHz, 2.6 GHz, 2.2 GHz, and 2.3 GHz, respectively, and all run with fully populated memory interfaces. The Intel machines are run with 2-way hyperthreading, e.g. with 96 threads for the Xeon Platinum Skylake.

The throughput results in Fig. 2 demonstrate the advancements of hardware during the last decade. In particular the increased width of vectorization, from 2 to 4 doubles with Sandy Bridge and from 4 to 8 doubles with Skylake, are clearly visible. Furthermore, the comparison between Sandy Bridge and the smaller Haswell system reveals the benefit of fused multiply-add (FMA) instructions and higher L1 cache bandwidth of the latter: For low polynomial degrees with a modest number of FMA instructions, Sandy Bridge with its higher frequency can approximately deliver the same performance as Haswell. As the polynomial degree is increased, the arithmetic work is increasingly dominated by FMAs in the sum factorization sweeps similar to (5) as shown in [50], and Haswell pulls ahead. Finally, while we observe a throughput of up to 5.7 billion DoFs/s on Skylake (with up to 1.35 TFlop/s for $k = 8$), we observe a relatively strong decrease of performance for polynomial degrees $k \geq 13$: This is because the vectorization across elements leads to an excessive size of the temporary data within sum factorization—here, a different vectorization strategy could lead to better results. However, we consider the polynomial degrees $3 \leq k \leq 8$ most interesting for practical simulations, where almost constant throughput in terms of DoFs/s is reached. This somewhat surprising result, given the expected $O(1/k)$ complexity of throughput for sum factorization, is because face integrals and memory access with an $O(1)$ complexity are dominant. Compared to our initial implementation in 2015, which achieved a throughput of 0.32 billion DoFs/s on Sandy Bridge with degree $k = 3$, the progress in software technologies allowed us to reach 1.02 billion DoFs/s on the same system. For Intel Skylake, where memory access is more important, the software progress of our project is more than 4×.

Figure 3 shows the throughput normalized by the number of cores for polynomial degree $k = 4$ over the different hardware generations. For operator evaluation with discontinuous elements and face integrals, approximately 200 floating point operations per unknown are involved with our optimized implementations [50]. At the same time, we must access at least 16 byte (read one double, write one double) plus some neighbors that are not cached, so the arithmetic intensity is around 8–12 Flop/Byte, close to the machine balance of the Skylake Xeon. This means that both memory bandwidth and arithmetic performance are relevant for performance (on one Skylake node, we measured memory throughput of around 160 GB/s, compared to the STREAM limit of 205 GB/s). Likewise, continuous elements evaluated on an affine mesh have seen a considerable increase in throughput per core (arithmetic intensity of 7 Flop/Byte). However, the improvement has been much more modest for continuous elements evaluated on curved elements. In this setting, separate

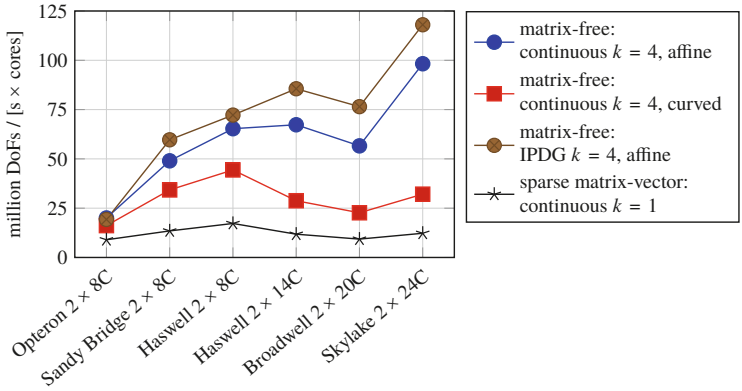


Fig. 3 Evolution of throughput of matrix-vector product per core (computed on a fully populated node and divided by the number of cores) with matrix-free evaluation with $k = 4$ versus a sparse matrix-vector product with continuous linear elements on various hardware

metric terms for all quadrature points and all elements are needed (as opposed to a single term per element in the affine mesh case), reducing the arithmetic intensity to around 1.2 Flop/Byte.[1]

Figure 3 also contains the evolution of performance of a sparse matrix-vector product for tri-linear continuous finite elements. The performance is much lower due to the aforementioned memory bandwidth limit, and has hardly improved per core on Skylake over the dated Opteron architecture. This illustrates the effect of the so-called memory wall. We emphasize that the sparse matrix-vector product for $k = 1$ is more than three times slower than even the matrix-free evaluation for $k = 4$ on curved elements. Hence, high-order methods with matrix-free implementations are faster per unknown on newer hardware, *in addition* to their higher accuracy.

3 Performance-Optimized Conjugate Gradient Methods

The developments of matrix-free implementations presented in the previous section result in a throughput for evaluation of the IPDG operator in Fig. 2 of up to 5.7 billion DoFs/s on Skylake. This is equivalent in time to the mere access of 4.5 doubles per DoF (either reading or writing). In other words, our developments have made the operator evaluation so fast that the matrix-vector product may no longer be the dominant operation in algorithms like the conjugate gradient (CG) method preconditioned by the diagonal, or Chebyshev smoothers. These algorithms

[1]The merged final coefficient tensor $\mathcal{J}_K^{-1}\mathcal{J}_K^{-\mathsf{T}}\det(\mathcal{J}_K)w_q$ is used for the present results, i.e., 6 doubles per quadrature point [29, 51].

www.dbooks.org

involve access to between 6 and 18 vectors for vector updates, the application of the diagonal preconditioner, and inner products. For optimal application performance it is therefore necessary to look into the access to vectors. As proposed in our work [51, 56, 65], merging the vector operations can improve throughput by up to a factor of two, and in particular for the DG case with cell-based loops which allow for a single pass through data [48, 56]. Fusion of different steps of a scheme has also been proposed for explicit time integrators in [14].

For the assessment of optimization opportunities on the algorithm level that goes beyond the matrix-vector product, we consider a high-order finite element benchmark problem suggested by the US exascale initiative "Center for Efficient Exascale Discretization" (CEED). The benchmark involves a continuous finite element discretization of the Laplacian (3), using matrix-free operator evaluation within a conjugate gradient solver preconditioned by the matrix diagonal. In this study, we consider the case BP5 [29], see also https://ceed.exascaleproject.org/bps/, which integrates the weak form (4) of polynomial degree k using a Gauss–Lobatto quadrature formula with $n_q^{\mathrm{1D}} = k + 1$ quadrature points on a cube with deformed elements. While this integration is not exact, it is the typical spectral element setup with an identity interpolation matrix $S_i = I$ in Eq. (5).

Figure 4 lists the contributors to run time for the plain conjugate gradient method preconditioned by the point-Jacobi method as a function of the problem size for the polynomial degree $k = 6$. Here, the metric terms $\mathcal{J}_K$ are computed on the fly from a tri-linear representation of the geometry. Three different performance regimes can be distinguished in the graph: To the left, there is not enough parallelism given the domain decomposition on 48 MPI ranks and batches of 8 elements due to vectorization—indeed, at least 85,000 DoFs are needed to saturate all cores and SIMD lanes. Furthermore, the synchronization barriers due to the inner products in the conjugate gradient method also lead to a slowdown. As the problem size and parallelism increase, the run times decrease significantly and reach a minimum for a problem size around one million DoFs. Here, all data involved in the algorithm

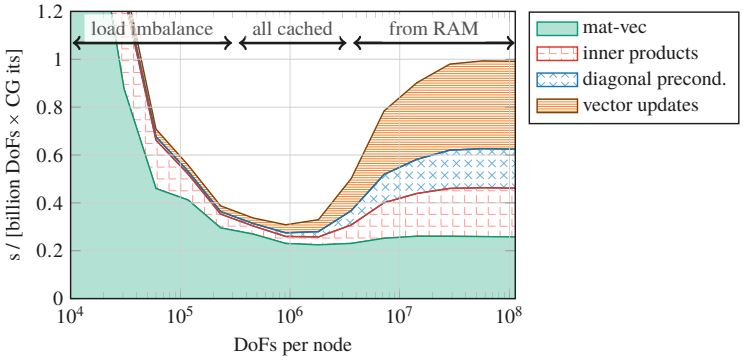


Fig. 4 Breakdown of times per CG iteration in CEED benchmark problem BP5 [29] for the plain conjugate gradient method with $k = 6$ on one node of dual-socket Intel Skylake

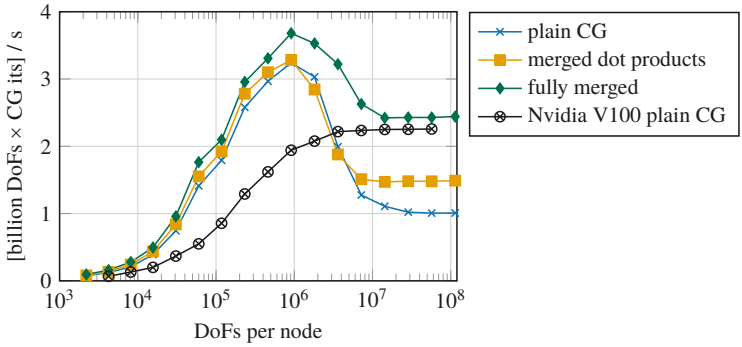


Fig. 5 Study of merged vector operations for conjugate gradient solver for the CEED benchmark problem BP5 [29] on one node of dual-socket Intel Skylake for $k = 6$

fits into the approximately 110 MB of L2+L3 cache on the processors. As the size is further increased, caches are exhausted and most data must be fetched from slow main memory. As a consequence, the run time of the solver increases significantly, and the vector updates, the diagonal preconditioner, and the inner products take a significant share. Note that all vector operations use the hardware optimally with a memory throughput of 205 GB/s.

In order to improve performance, we have therefore developed conjugate gradient implementations with merged vector operations by loop fusion. Figure 5 compares three variants of the conjugate gradient solver: the plain conjugate gradient method runs all vector operations through high-level vector interfaces with separate loops for addition and inner products. In the "merged dot products", we have merged the dot product $p^{\mathsf{T}}Ap$ following the matrix-vector product into the loop over the elements, and merged the vector updates to the residual and solution with the dot product for $r^{\mathsf{T}}P^{-1}r$. Here, r denotes the residual vector, p the search direction of the conjugate gradient method, A the matrix operator (represented in a matrix-free way), and P^{-1} the diagonal preconditioner. However, the improvements with this algorithm are relatively modest.

Much more performance can be gained by creating a conjugate gradient variant we call "fully merged": Here, each CG iteration performs a single loop through all vector entries and ideally reads 5 vectors (solution, residual, search direction, temporary vector to hold the matrix-vector product, and diagonal of preconditioner) and writes four (solution, residual, search direction, temporary vector). All vector updates of the previous CG iteration are scheduled before the matrix-vector product and all inner products are scheduled after the matrix-vector product. The vector operations are interleaved with the loop over elements, ensuring that dependencies due to the access pattern of the loop and the MPI communication are fulfilled (this leads to slightly more access in practice). This approach applies the preconditioner several times with partial sums to construct the inner products with a single `MPI_Allreduce`, trading some local computations for the decreased memory

www.dbooks.org

access. Of course, fusing the preconditioner into the loop assumes that it is both cheap to apply and does not involve long-range coupling between the DoFs. The results in Fig. 5 show that performance in the saturated limit, i.e., for large sizes beyond 10^7 DoFs, is 2.5 times faster than with the plain CG iteration. Interestingly, this also improves performance for the sizes fitting into caches, which is due to less synchronization and reducing access to the slower L3 cache.

To put the performance of the fully merged case on Intel Skylake into perspective, we compare with executing the plain CG method on an Nvidia V100 GPU using the implementation from [51, 57]: even though the GPU runs with around 700 GB/s of memory throughput, the performance is higher on Intel Skylake with only 200 GB/s from RAM memory because the merged loops significantly increase data locality. Furthermore, on the GPU we do not compute the metric terms on the fly, but load a precomputed tensor $\mathcal{J}_K^{-1}\mathcal{J}_K^{-\mathsf{T}}\det(\mathcal{J}_K)w_q$ which is faster due to reduced register pressure, see also the analysis for BP5 in [71]. We also note that the GPU results with our implementation are faster than an implementation with the OCCA library described in [29] with up to 0.6 billion DoFs/s on a V100 of the Summit supercomputer. The reason is that our implementation uses a continuous finite element storage that does not duplicate the unknowns at shared vertices, edges and faces, which reduces the memory access by about a factor of two. Furthermore, the results from [29] involve a separate gather/scatter step with additional memory transfer to enforce continuity, while this is part of the operator evaluation within a single loop in our code.

Figure 6 lists the achieved throughput with a fully merged conjugate gradient solver for polynomial degrees $k = 2, \ldots, 8$, the most interesting regime for our solvers. We use a tri-linear representation of geometry and compute the geometric

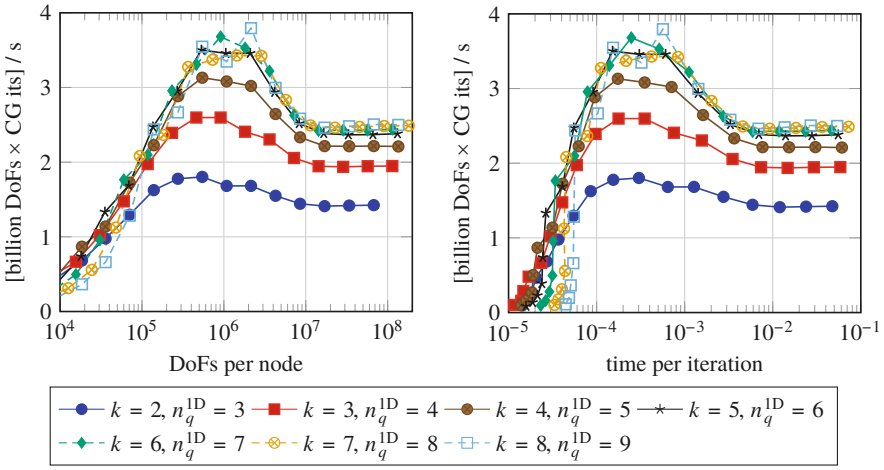


Fig. 6 Throughput of the CEED benchmark problem BP5 [29] on one node of dual-socket Intel Skylake for $k = 2, \ldots, 8$ with fully merged conjugate gradient solver

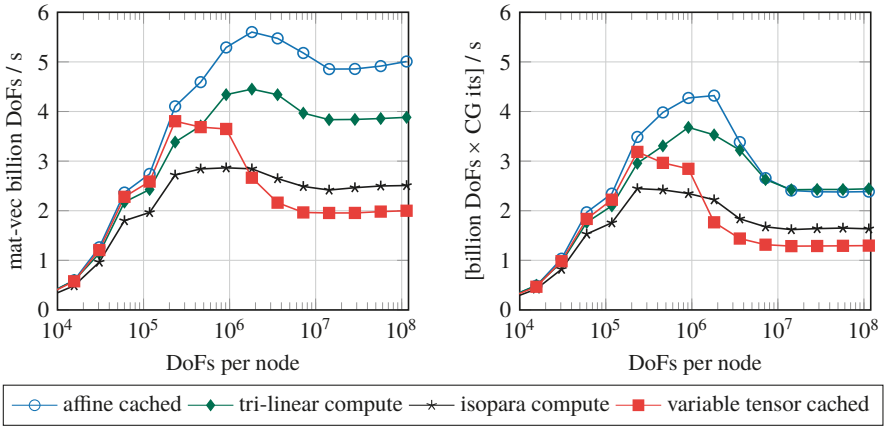


Fig. 7 Study of geometry representation for the CEED benchmark problem BP5 [29] on one node of dual-socket Intel Skylake with $k = 6$ for matrix-vector product only (left) and with fully merged vector operations in the conjugate gradient solver (right)

factors on the fly. Throughput is somewhat lower for quadratic and cubic elements because the geometry data located in the vertices is still noticable.

The results in Fig. 3 motivate the analysis of the representation of the geometry in the matrix-vector product, with results presented in Fig. 7. The figure lists both the throughput of the matrix-vector product in the left panel and the throughput of the complete CG iteration with merged vector operations. Highest performance is obtained for the affine mesh case where our implementation can compress the memory access of the Jacobian. While this case is excluded from the CEED BP5 specification that requires a deformed geometry [29], it is an interesting baseline to compare against. Using separate tensors for each quadrature point, "variable tensor cached", is equally fast as the affine case as long as data fits into caches. However, performance drops once the big geometric arrays must be fetched from main memory. For the case the geometry is computed on the fly from a tri-linear representation of the mesh, i.e., the vertices, the matrix-vector product is slower than the affine variant. For the conjugate gradient solver, however, we observe that the two reach essentially the same performance for five million and more DoFs, as they are both limited by the memory bandwidth from vector access. The "tri-linear compute" case involves a higher Flop/s rate with almost 700 GFlop/s, as compared to the throughput of 330 GFlop/s for the affine mesh case. This means that the merged vector operations allow us to fit additional computations behind the unavoidable memory transfer *without* affecting application performance. Finally, an isoparametric representation of the geometry (labeled "isopara compute" in Fig. 7) can also be computed on the fly by sum factorization from a kth degree polynomial [50]. While this case is obviously slower than the precomputed variable-tensor case from caches, it leverages higher performance when data must be fetched from main memory.

www.dbooks.org

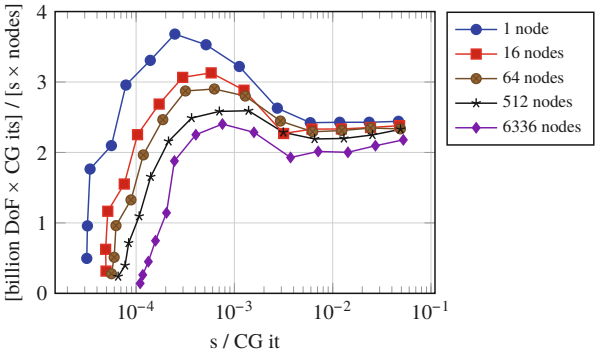


Fig. 8 Throughput of the CEED benchmark problem BP5 [29] with $k = 6$ on up to the full SuperMUC-NG machine. Throughput normalized per node

The tri-linear and isoparametric cases are not equivalent, as only the latter represents higher order curved boundaries. Intermediate polynomial degrees for the geometry are conceivable, which would land between the two in terms of application throughput. To combine the higher performance of the former, we plan to investigate the tradeoffs in more detail in the future, e.g. by using a k-degree representation on a single layer of elements at the boundary and a tri-quadratic representation in the domain's interior.

Finally, Fig. 8 shows the weak scaling of the BP5 benchmark problem up to the full size of the SuperMUC-NG machine with 6336 nodes and 304,128 cores. The data is normalized by reporting the number of DoF per node, so ideal weak scaling would correspond to coinciding lines. While the saturated performance is scaling well, giving a sustained performance of up to 4.4 PFlop/s,[2] most of the in-cache performance advantage is lost due to the communication latency over MPI, see also [62] for limits with MPI in PDE solvers. Defining the strong scaling limit as the point where throughput reduces to 80% of saturated performance [29], it is reached for wall times of 56 µs on 1 node. On 512 nodes, the strong scaling limit is already around 180 µs, whereas it is 245 µs on the full SuperMUC-NG machine. Note that even though most optimizations presented in this section have addressed the node-level performance, we have also considered the strong scaling in our work—indeed, the strong scaling on SuperMUC-NG is excellent with a limit around 5 times lower than the BlueGene-Q results presented in [29].

[2]The LINPACK performance of SuperMUC-NG according to the top500 list is 19.4 PFlop/s. Considering that we use an iterative solver for PDE with optimization of throughput, this is an extremely good value.

4 Geometric Multigrid Methods in Distributed Environments

Multigrid methods are efficient solvers for the linear systems arising from the discretization of elliptic problems, see [30] for a recent efficiency evaluation and [35] for a projection of elliptic solver performance to the exascale setting. They apply simple iterative schemes called smoothers on a hierarchy of coarser problem representations. On each level of the hierarchy, the smoothers address the high-frequency content of the solution by smoothening the error. On a sufficiently coarse level with a small number of unknowns, a direct solver can be applied. The multigrid algorithm can be realized by a V-cycle as illustrated in Fig. 9 or some related cycle (W-cycle or F-cycle). In the matrix-free high-order finite element context, variants of the Chebyshev iteration around a simple additive scheme, such as point-Jacobi or approximate block-Jacobi with some rank-d approximation of the cell matrix, are state of the art. The results in this section are based on this selection. Overlapping Schwarz schemes are a new development detailed in Sect. 5 below.

In terms of finding the coarser representations for the multigrid hierarchy, high-order finite element and discontinuous Galerkin methods permit a range of options. The hierarchy can both be constructed by coarser meshes (h-multigrid), by lowering the polynomial degree (p-multigrid), by a discontinuous-continuous transfer as well as algebraically based on the matrix entries only (algebraic multigrid). The latter do not fit into a matrix-free context, since they explicitly rely on a sparse matrix and also often are not robust enough as the degree increases. As it has been shown by the work [70], scalability to the largest supercomputers is much more favorable if knowledge about coarsening by a mesh can be provided. In other words, geometric multigrid is to be preferred over algebraic multigrid in case there is such structure in the problem.

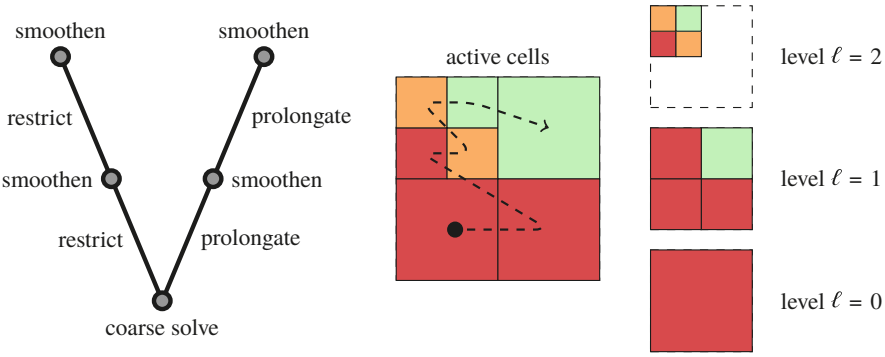


Fig. 9 Illustration of multigrid V-cycle with smoothing on each level and restriction/prolongation between the levels (left) and exemplary partitioning of a grid with adaptive refinement partitioned among 3 processors. The partitioning of the active cells is shown in the mid panel and on the various multigrid levels on the right panel

www.dbooks.org

For these reasons, we have developed a comprehensive geometric multigrid framework with deal.II. In [27], a hybrid multigrid solver with all possibilities of h-, p-, and algebraic coarsening has been combined in a flexible framework, with the possibility to perform an additional c-transfer from discontinuous to continuous function spaces for the DG case. In terms of the h-MG method on adaptive meshes, the deal.II library implements the local smoothing algorithm [9, 36, 37] where smoothing is done level by level. Our work [15] developed a communication-efficient coarsening strategy for this setup, at the cost of a load imbalance for smoothing on the multigrid levels with adaptively refined meshes. The tradeoffs in this choice and the associated costs have been quantified by a performance model in [15].

Figure 10 shows the results of two strong scaling experiments of the multigrid V-cycle with the h-multigrid infrastructure of the deal.II library. The uniform grid and a typical adaptively refined case are compared for the same problem size of 137 million and 46 billion DoFs, respectively, see [15] for details on the experiment. Differences in run time are primarily due to the load imbalance for the level operations. The results demonstrate optimal parallel scaling of both the uniform and adaptively refined cases down to around 10^{-2} s, with a slightly better strong scaling of the adaptive case due to the slower baseline. This performance barrier—typical for strong scaling of multigrid schemes in general—can be explained by the specific type of global communication in this algorithm: from the fine mesh level with many unknowns distributed among a large number of cores, we transfer residuals to coarser meshes with restriction operators until the coarse grid solver is either run on a single core or with few cores in a tightly coupled manner. Then, the coarse-grid corrections are broadcast during prolongation, involving all processors again. The communication pattern of a multigrid V-cycle thus relates to a tree-based

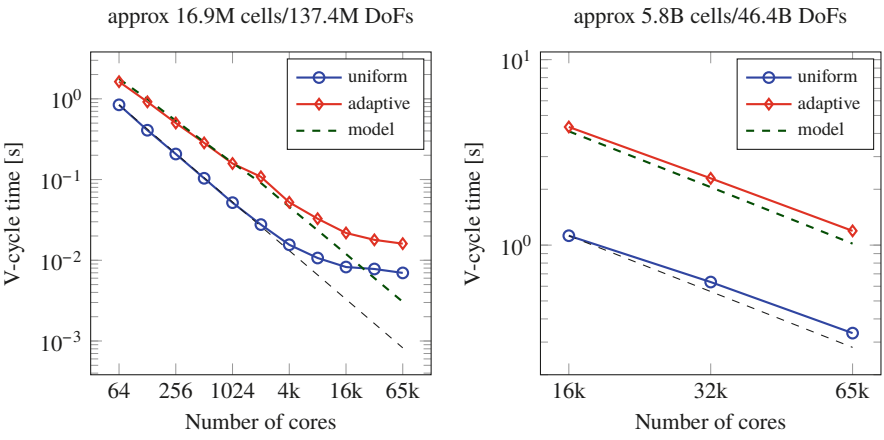


Fig. 10 Strong scaling of geometric multigrid V-cycle for 3D Laplacian on uniform and adaptively refined mesh using continuous Q_2 elements with matrix-free evaluation on up to 4096 nodes (64k cores) of 2×8 core Intel Sandy Bridge (SuperMUC phase 1). Adapted from [15]

implementation of `MPI_Allreduce`, with the difference that the communication tree is induced by the grid and substantial operations, namely smoothing and level transfer, are intermixed with the communication. In this particular case, nine matrix-vector products with nearest-neighbor communication are performed per level (eight in the smoother and one for the residual before restriction). In addition, two vertical nearest-neighbor exchange operations are done in restriction and prolongation. A typical matrix-vector product with up to 26 neighbors takes around 10^{-4} s on the chosen Intel Sandy Bridge system when run on a few thousands of nodes [52]. When done on seven levels plus the coarse mesh for the uniformly refined 137 million DoFs case, the expected saturated limit of around 8 ms is exactly seen in the figure. On the newer SuperMUC-NG machine, a latency barrier per V-cycle of around 2–4 ms per V-cycle has been measured, depending on the number of matrix-vector products for the level smoothers. This limit is attractive compared to alternative solvers for elliptic problems such as the fast multipole method or the fast Fourier transform [30, 35].

Multigrid schemes are at the heart of incompressible flow solvers through the pressure Poisson equation, as detailed in Sect. 6 below. Applications of matrix-free geometric multigrid to continuum mechanics were presented in [18] and to electronic calculations with sparse multivectors in [16, 17].

As an example of the large-scale suitability of the developed multigrid framework, Fig. 11 shows two scaling experiments on the SuperMUC-NG supercomputer with up to 304,128 cores of Intel Skylake. Black dashed lines denote ideal strong scaling along a line and weak scaling with a factor of 8 between the lines. The computational domain is a cube meshed by hexahedral elements, using the affine mesh code path for matrix-free algorithms discussed in Sect. 2. A consistent Gaussian quadrature with $n_q^{1D} = k + 1$ points is chosen. We run a conjugate gradient solver to a relative tolerance of 10^{-3} compared to the initial unpreconditioned residual. This setup is motivated by applications where a very good initial guess is already

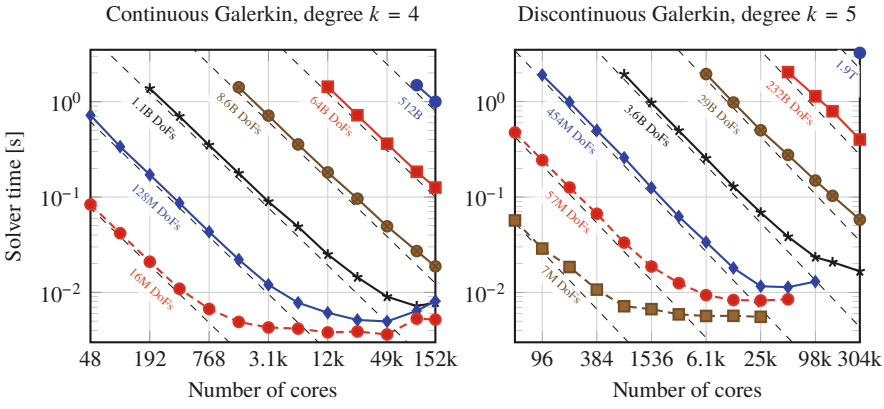


Fig. 11 Multigrid strong scaling analysis for tolerance 10^{-3} with 2 CG iterations

www.dbooks.org

available, e.g. by extrapolation of solutions from the old time step [22, 45], and only a correction is needed. More accurate solves are obtained by tighter tolerances or by full multigrid setups [51]. The multigrid V-cycle is run in single precision to increase throughput, together with a double precision correction through the outer CG solver. This setup has been shown in [51] to increase throughput by around $1.8\times$ without affecting the multigrid convergence.

In the left panel of Fig. 11, we present results for a continuous Galerkin discretization with a polynomial degree $k = 4$. A pure geometric coarsening down to a single mesh element is used. A Chebyshev iteration of degree five based on the matrix diagonal, i.e., point Jacobi, is used on all levels for pre- and post-smoothing. The maximal eigenvalue $\tilde{\lambda}_{\max}$ is estimated by 15 iterations of a conjugate gradient solver and the Chebyshev parameters are set to smoothen in a range $[0.06\tilde{\lambda}_{\max}, 1.2\tilde{\lambda}_{\max}]$. As a coarse solver, we use a Chebyshev iteration with the degree set to reduce the residual by 10^3 in terms of the Chebyshev a-priori error estimate [74]. We observe ideal weak scaling and strong scaling to around 10^{-2} s. More importantly, the absolute run time is excellent: For instance, the 8.6 billion DoF case on 1536 cores is solved in 1.4 s, i.e., 4.0 million DoFs are solved per core per second.

The right panel of Fig. 11 shows the result for multigrid applied to an IPDG discretization with $k = 5$. Here, we use a transfer from the discontinuous space to the associated continuous finite element space with $k = 5$ on the finest mesh level (see [3] for the theoretical background and [27] for the multigrid context) and then progress by h-coarsening to a single element. On the DG level, we use a Chebyshev smoother around a block-Jacobi method, with the block-Jacobi problems inverted by the fast diagonalization method [58]. On all continuous finite element levels, a Chebyshev iteration around the point-Jacobi method is used. The degree of the Chebyshev polynomial is six. This solver setup achieves a multigrid convergence rate of about 0.025, i.e., reduces the residual by 3 orders of magnitude with two V-cycles. If used in a full multigrid setting [51], a single V-cycle on the finest level would suffice to solve the problem to discretization accuracy. Merged vector operations with a Hermite-like basis for the Chebyshev iteration are used according to [56]. The final application performance of the largest computation on 1.9 trillion DoFs is 5.9 PFlop/s, with 5.6 PFlop/s done in single precision and 0.27 PFlop/s in double precision. The limiting factor is mostly memory transfer, however, with an application throughput of around 175 GB/s per node (the STREAM limit of one node is 205 GB/s).

5 Fast Tensor Product Schwarz Smoothers

In Sect. 4, we have discussed a scalable implementation of geometric multigrid methods, obtaining an efficient solver in the sense of cost per iteration. It employs the matrix-free operator implementation from Sect. 2 in order to reduce the computational cost for residuals and grid transfer. The missing building block for our cost

model in Eq. (1) is an efficient implementation (in terms of computational cost per DoF) of an efficient smoother (in terms of number of multigrid iterations).

The main challenge consists of finding preconditioners whose cost is similar to operator evaluation. So far, we have discussed Chebyshev smoothers, which can be implemented matrix-free in a straight-forward fashion. Alas, their performance is not robust for higher order elements. Likewise, from an arithmetic cost point of view sparse matrices can be competitive at most for moderate polynomial degrees $k = 2, 3$ [55] or when done via auxiliary spaces of linear elements on a subdivided grid using some matrix-based preconditioner. However, Fig. 3 shows that even sparse matrices for linear elements are up to 10 times slower than the matrix-free operator evaluation. It seems that only the two SPPEXA projects ExaDUNE and ExaDG have addressed this question in [6, 76]. While [6] focuses on iterative solution of cell problems for multigrid smoothing, we consider domain decomposition based smoothers in the form of multilevel additive and multiplicative Schwarz methods based on low-rank tensor approximations. They consist of a subdivision of the mesh on each level into small subdomains consisting either of a single cell, or of the patch of cells sharing a common vertex. On each of these subdomains, local finite element problems are solved. Comparing with operator application, these smoothers share the structural property of evaluation of local operators on mesh cells or on a patch of cells. They differ by the fact that the smoothers involve local inverses instead of local forward operators, and that these local inverses in general are not amenable to a tensor decomposition like sum factorization. There is one exception though, namely separable differential operators. In d dimensions these can be written in the form

$$L = I_d \otimes \cdots \otimes I_2 \otimes L_1 + \cdots + L_d \otimes I_{d-1} \otimes \cdots \otimes I_1, \tag{6}$$

where L_k are one-dimensional differential operators and I_k are identity operators in directions $k = 1, \ldots, d$. This representation transfers to finite element operators with tensor product shape functions in a straight-forward way, reading I_k as one-dimensional mass matrices M_k.

Due to [58], the inverse of L can be represented as the product

$$L^{-1} = Q \Lambda^{-1} Q^{\mathsf{T}}, \tag{7}$$

with the diagonal matrix $\Lambda = I_d \otimes \cdots \otimes I_2 \otimes \Lambda_1 + \cdots + \Lambda_d \otimes I_{d-1} \otimes \cdots \otimes I_1$, where I_k denote identity matrices, and a rank-1 decomposition $Q = Q_d \otimes \cdots \otimes Q_1$. The tensor factors are obtained by solving d generalized eigenvalue problems

$$\begin{aligned} \Lambda_k &= Q_k^{\mathsf{T}} L_k Q_k, \\ I_k &= Q_k^{\mathsf{T}} M_k Q_k, \qquad k = 1, \ldots, d. \end{aligned} \tag{8}$$

Thus, the computational effort for computing the inverse has been reduced from $O(k^{3d})$ to $O(dk^3)$ and for the application of local solvers from $O(k^{2d})$ to $O(dk^{d+1})$

www.dbooks.org

by exploiting sum factorization. Based on this technique, we have implemented a geometric multigrid method in [76] based on earlier work in [37–39].

5.1 The Laplacian on Cartesian Meshes

In order to test our concept and to obtain a performance baseline for more complicated cases, we first attend to the case where the decomposition described above can be applied in a straightforward way, namely the additive Schwarz method with subdomains equal to mesh cells. As Table 1 shows, it yields an efficient preconditioner with less than 25 conjugate gradient steps for a gain of accuracy of 10^8. While it is uniform in the mesh level, it is not uniform in the polynomial degree due to the increasing penalty parameter of the interior penalty method. The computational effort for a smoothing step based on local solvers in the form (7) is below the effort for a matrix-free operator application for polynomial degrees between 3 and 15 in three dimensions because it only involves operations on cells. The setup time for computing Q and Λ is even less. Thus, in the context of the performance analysis of the conjugate gradient method in Sect. 2, it barely adds to the cost per iteration step, but reduces the number of matrix-vector products when comparing to the accumulated numbers within a Chebyshev/point Jacobi method, and almost independently of polynomial degree.

In view of application to incompressible flow, we also study vertex patches as typical subdomains for smoothing. First, we observe that a regular vertex patch with 2^d cells attached to a vertex inherits the low-rank tensor product structure from its cells, possibly after renumbering due to changes in orientation. Thus, we can apply the same method as on a single cell, resulting effectively in a factor 2^d in the complexity estimates above. Patches around vertices with irregular topology like 3 or 5 cells in two dimensions do not possess a tensor product structure. Fortunately,

Table 1 Fractional CG iterations, preconditioned by h-MG with additive Schwarz smoother on cells

Levels	Convergence steps			
2D	$k = 3$	$k = 4$	$k = 7$	$k = 10$
7	14.5	14.3	18.8	20.9
8	14.5	14.3	18.8	20.9
9	14.5	14.3	18.8	20.9
10	14.5	14.3	18.8	20.9
3D	$k = 3$	$k = 4$	$k = 7$	$k = 10$
3	16.7	16.8	22.0	24.5
4	17.1	17.0	22.0	24.5
5	17.2	17.0	22.1	24.6
6	17.1	17.0	22.1	24.7

Relative solver tolerance of 10^{-8} and relaxation parameter $\widehat{\omega} = 0.7$

on meshes obtained by refinement of a coarse mesh, they are all determined by irregularities of the coarse mesh and thus small in number.

Vertex patches lead to overlapping decompositions with overlap of at least 4 and 8 in two and three dimensions, respectively. From the analysis of Schwarz methods, it becomes clear that a multiplicative method is required for highest multigrid convergence rates. In order to parallelize such a smoother and to avoid race conditions, mesh cells are colorized, that is, they are separated into "colors" such that patches of the same "color" do not share any common face or cell. As a consequence, the multiplicative method coincides with an additive method within each color, such that we can execute the local solvers in parallel within each color, and the colors sequentially. Typical convergence results for the Laplacian are reported in Table 2, suggesting that this scheme is almost a direct solver.

The vertex patch has 4 and 8 times as many unknowns as a single mesh cell in two and three dimensions, respectively. Thus, the effort for a smoothing step with 16 colors and the optimizations described above turns out to be about 20 to 24 times the effort of a matrix-free operator application, measured over polynomial degrees from 3 to 15. This seems excessive at first glance, but it must be kept in mind that the Chebyshev smoother of degree 6 used in Fig. 11 also involves 12 matrix-vector products for pre- and post-smooting. Futhermore, the current scheme comes with a reduction of the number of steps by a factor 10 compared to the additive cell smoother for the Laplacian, which makes it almost competitive [76]. Finally, the iteration counts are independent of the polynomial degree, making the scheme attractive for higher degrees. Moreover, we point out that this smoother also allows for the solution of a Stokes problem in four iteration steps [39].

Table 2 Fractional GMRES iterations, preconditioned by h-MG with multiplicative Schwarz smoothers on vertex patches

Levels	Convergence steps				Colors	Levels	Convergence steps				Colors
2D	$k=3$	$k=4$	$k=7$	$k=10$		2D	$k=3$	$k=4$	$k=7$	$k=10$	
7	2.5	2.5	2.1	2.1	8	7	2.9	2.9	2.6	2.5	17
8	2.5	2.5	2.1	2.0	8	8	2.9	2.9	2.6	2.5	17
9	2.5	2.4	2.1	2.0	8	9	2.9	2.9	2.6	2.5	17
10	2.5	2.4	2.0	2.0	8	10	2.9	2.9	2.6	2.4	17
3D	$k=3$	$k=4$	$k=7$	$k=10$		3D	$k=3$	$k=4$	$k=7$	$k=10$	
3	2.4	2.5	2.1	1.8	16	3	2.6	2.7	2.4	2.4	35
4	2.4	2.5	2.1	1.9	16	4	2.8	2.8	2.5	2.4	49
5	2.4	2.5	2.1	1.9	16	5	2.8	2.8	2.5	2.4	51
6	2.4	2.5	2.1	1.9	16	6	2.8	2.8	2.5	2.4	52

Based on minimal coloring (left) and graph coloring (right) with a relative solver tolerance of 10^{-8}

www.dbooks.org

5.2 General Geometry

As soon as the mesh cells are not Cartesian anymore, the special structure of separable operators in (6) is lost and the inverse cannot be computed according to (7). In this case, we have two options: solving the local problems iteratively, as in [6], or approximately. A possible approximation which recovers the situation of the previous subsection consists of replacing a non-Cartesian mesh cell by an approximating (hyper-)rectangle, then inverting the separable differential operator on the rectangle (omitting the prefix hyper from here on).

Such a surrogate rectangle can be obtained from the following procedure: first, we compute the arc length of all edges. From these, we obtain the length of the rectangle in each of its natural directions by averaging over all parallel edges (in a topological sense). Thus, the geometry of the rectangle is determined up to its position and orientation in space. Given the fact that the Laplacian is invariant under translation and rotation, these do not matter and we can choose a rectangle centered at the origin with edges parallel to the coordinate directions. Different differential operators may require different approximations here.

The convergence theory of Schwarz methods allows for inexact local solvers as long as they are spectrally equivalent. Naturally, the deviation from exactness enters into the convergence speed of the method. Additionally, inexact local solvers can amplify the solution, such that a smaller relaxation parameter may be necessary. This is exhibited in Table 3, where we compare the efficiency of multigrid with exact local solvers and the method with surrogate rectangles as described above. We see that a reduction of the relaxation parameter $\widehat{\omega} = 0.7$ for exact local solvers to $\widehat{\omega} = 0.49$ is necessary for robust convergence. We point out though, that while the inexact methods need more iteration steps, they are much faster than exact inverses, since they use the Kronecker representation (7) of the approximate inverse. For instance, the setup cost is 3000 times higher, with a growing gap for higher polynomial degrees.

Table 3 Fractional CG iterations with addditive cell-based Schwarz smoothers, exact as well as inexact local solution with varying damping factors $\hat{\omega}$

Levels	Convergence steps to 10^{-8}					
2D	*exact* ($\hat{\omega} = 0.7$)	$\hat{\omega} = 0.35$	$\hat{\omega} = 0.42$	$\hat{\omega} = 0.49$	$\hat{\omega} = 0.56$	$\hat{\omega} = 0.63$
4	17.8	28.4	24.8	24.3	30.8	>100
5	17.3	27.1	23.9	23.8	40.7	>100
6	17.2	26.8	23.7	23.9	58.1	>100
3D	*exact* ($\hat{\omega} = 0.7$)	$\hat{\omega} = 0.35$	$\hat{\omega} = 0.42$	$\hat{\omega} = 0.49$	$\hat{\omega} = 0.56$	$\hat{\omega} = 0.63$
2	20.6	31.8	28.5	25.8	25.0	28.5
3	20.6	33.3	29.1	26.5	27.4	74.8
4	20.6	32.4	28.6	26.6	47.0	>100

Two pre- and post-smoothing steps are used, respectively, and the polynomial degree is $k = 4$

5.3 Linear Elasticitiy

In order to provide an outlook on how to apply this concept to more general problems, we consider linear elasticity, namely the Lamé-Navier equations, with the bilinear form

$$a(u,v) = 2\mu\big(\varepsilon(u),\varepsilon(v)\big) + \lambda\big(\nabla\cdot u, \nabla\cdot v\big). \tag{9}$$

Here, $\varepsilon(u) = \frac{1}{2}(\nabla u + \nabla u^{\mathsf{T}})$ is the strain tensor of the displacement field u and $\big(\cdot,\cdot\big)$ denote the appropriate DG discretization with interior penalty terms.

Consider a Cartesian vertex patch, that is, a patch with all faces aligned with the coordinate planes and with tensor product shape functions on each cell. As before, let M_k be the one-dimensional mass matrix in direction k and L_k the matrix representing the Laplacian including all face terms introduced by the interior penalty formulation. Furthermore, let G_k be the matrix associated to the first derivative, again including the DG interface terms which arise in products of the form $G_k^{\mathsf{T}}\otimes G_l$. With these notions and the three-dimensional Laplacian

$$L = M_3\otimes M_2\otimes L_1 + M_3\otimes L_2\otimes M_1 + L_3\otimes M_2\otimes M_1, \tag{10}$$

we can write the bilinear form $a(.,.)$ on the patch in matrix form

$$\begin{aligned} A_p = \mu &\begin{bmatrix} L + M_3\otimes M_2\otimes L_1 & M_3\otimes G_2^{\mathsf{T}}\otimes G_1 & G_3^{\mathsf{T}}\otimes M_2\otimes G_1 \\ M_3\otimes G_2\otimes G_1^{\mathsf{T}} & L + M_3\otimes L_2\otimes M_1 & G_3^{\mathsf{T}}\otimes G_2\otimes M_1 \\ G_3\otimes M_2\otimes G_1^{\mathsf{T}} & G_3\otimes G_2^{\mathsf{T}}\otimes M_1 & L + L_3\otimes M_2\otimes M_1 \end{bmatrix} \\ + \lambda &\begin{bmatrix} M_3\otimes M_2\otimes L_1 & M_3\otimes G_2\otimes G_1^{\mathsf{T}} & G_3\otimes M_2\otimes G_1^{\mathsf{T}} \\ M_3\otimes G_2^{\mathsf{T}}\otimes G_1 & M_3\otimes L_2\otimes M_1 & G_3\otimes G_2^{\mathsf{T}}\otimes M_1 \\ G_3^{\mathsf{T}}\otimes M_2\otimes G_1 & G_3^{\mathsf{T}}\otimes G_2\otimes M_1 & L_3\otimes M_2\otimes M_1 \end{bmatrix} \end{aligned} \tag{11}$$

Clearly, this matrix lacks the simple structure of Kronecker products we employed in the previous subsections. Nevertheless, we have Korn's inequality [10], and thus the block diagonal of the left matrix is spectrally equivalent to the matrix itself. Consequently, we expect that

$$\tilde{A}_p = \mu \begin{bmatrix} L + M_3\otimes M_2\otimes L_1 & & \\ & L + M_3\otimes L_2\otimes M_1 & \\ & & L + L_3\otimes M_2\otimes M_1 \end{bmatrix}, \tag{12}$$

which has the desired Kronecker product structure, is a good local solver. Indeed, Table 4 confirms this expectation. Iteration counts remain almost constant over a

www.dbooks.org

Table 4 Solver performance depending on level and polynomial degree k

Levels	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$
3	–	–	–	–	–	–	–	4.0	4.1
4	–	–	–	3.7	3.9	3.9	4.0	4.1	4.1
5	–	3.7	3.7	3.7	3.8	3.9	3.9	3.9	3.9
6	5.1	3.7	3.8	3.6	3.8	3.9	3.9	3.9	3.9
7	5.2	3.8	3.9	3.7	3.7	3.8	3.7	3.8	3.8
8	5.5	3.9	3.9	3.8	3.8	3.7	3.8	–	–
9	5.4	3.9	4.0	–	–	–	–	–	–

CG iterations to reduce the residual by 10^8 preconditioned by h-MG with multiplicative vertex patch smoother and approximate local solvers $\tilde{A}_p^{-1}$. Only levels with 10^4 to 10^7 degrees of freedom are shown. $\mu = 1$, $\lambda = 1$ and the coarse grid consists of 2×2 cells

Table 5 Solver performance depending on Lamé parameters μ and λ

	(μ, λ)					
Levels	(100, 1)	(10, 1)	(1, 1)	(1, 5)	(1, 10)	(1, 25)
6	3.4	3.4	3.6	6.3	19.5	>200
7	3.6	3.6	3.7	6.2	19.9	>200
8	3.7	3.7	3.8	6.0	20.2	>200
9	3.8	3.8	3.9	5.9	20.2	>200
10	3.8	3.8	3.9	5.9	20.3	>200
11	3.8	3.8	3.9	5.8	19.9	>200

CG iterations to reduce the residual by 10^8 preconditioned by h-MG with block-diagonal smoother. Shape functions of degree $k = 4$ are used. The coarse grid consists of 2×2 cells

wide range of mesh levels and polynomial degrees. Comparing to Table 2, we lose less than a factor two, typically requiring 4 steps instead of 3.

While Korn's inequality helped us with the left matrix in (11), the matrix corresponding to the "grad-div" term in the Lamé–Navier equations has a nontrivial kernel and thus its inverse cannot be approximated by a block diagonal. We confirm this in Table 5. After augmenting $\tilde{A}_p$ by the diagonal terms of the grad-div matrix, we vary μ and λ. As expected, iteration counts increase when $\lambda \gg \mu$ to the point, where the method becomes infeasible.

The case $\lambda \gg \mu$ corresponds to an almost incompressible material. Thus, this behavior has to be addressed from two sides. First, the discretization must be suitable [34]. Then, the local solvers must be able to reduce the divergence sufficiently. Here, we have to find ways to implement a smoother like in [39] in an efficient way. Its structure prevents us from utilizing the tensor product techniques, namely the fast diagonalization method, used so far.

As an outlook, we describe a solution approach for two dimensions, which has been developed in a recent bachelor's thesis [64]. The diagonal blocks of the matrix A_p are

$$A_1 = (2\mu+\lambda)\, M_2 \otimes L_1 + \mu L_2 \otimes M_1, \quad A_2 = \mu M_2 \otimes L_1 + (2\mu+\lambda)\, L_2 \otimes M_1. \tag{13}$$

Both A_1 and A_2 admit a fast diagonalization, for instance

$$A_2^{-1} = \big(Q_2 \otimes Q_1\big)\big(I_2 \otimes \Lambda_1 + \Lambda_2 \otimes I_1\big)^{-1}\big(Q_2 \otimes Q_1\big)^{\mathsf{T}}. \tag{14}$$

Given the off-diagonal block $B = \mu G_2^{\mathsf{T}} \otimes G_1 + \lambda G_2 \otimes G_1^{\mathsf{T}}$, the Schur complement of A_p is

$$S = A_1 - B^{\mathsf{T}} A_2^{-1} B. \tag{15}$$

While this is not a sum of Kronecker products, Kronecker singular value decomposition (KSVD), see [72, 73], can be utilized to construct an approximation of the Schur complement which is fast diagonalizable. We proceed as follows:

A.1 compute the fast diagonalizations of A_1 and A_2
A.2 compute the rank-ρ_Λ KSVD of the inverse diagonal matrix in Eq. 14

$$\big(I \otimes \Lambda^{(1)} + \Lambda^{(2)} \otimes I\big)^{-1} \approx \sum_{i=1}^{\rho_\Lambda} C_i \otimes D_i \tag{16}$$

A.3 compute the rank-2 KSVD

$$\widehat{S} := E_1 \otimes F_1 + E_2 \otimes F_2 \approx \widetilde{S} \tag{17}$$

of the approximate Schur complement

$$\widetilde{S} := A_1 - B^{\mathsf{T}} \left[\sum_{i=1}^{\rho_\Lambda} Q_2 C_i^{-1} Q_2^{\mathsf{T}} \otimes Q_1 D_i^{-1} Q_1^{\mathsf{T}}\right] B \tag{18}$$

A.4 compute the fast diagonalization of $\widehat{S}$.

Then, Gaussian block elimination provides an approximate inverse

$$A_p^{-1} \approx \begin{bmatrix} I & -A_1^{-1}B \\ 0 & I \end{bmatrix} \begin{bmatrix} A_1^{-1} & 0 \\ 0 & \widehat{S}^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -B^{\mathsf{T}} A_1^{-1} & I \end{bmatrix}. \tag{19}$$

www.dbooks.org

Implementation and evaluation of these smoothers are still work in progress, but the thesis [64] suggests fast and robust convergence at least in a finite difference context.

The take-home message from this section is that an efficient approximate solution of the local problems in Schwarz smoothers is possible using low-rank tensor representations and can be achieved with effort similar to a matrix-free operator application in the best case. Finding such low-rank representations is nevertheless highly dependent on the differential equation and geometry. Further investigation will be directed in particular at dealing with the grad-div operator.

6 High-performance Simulations of Incompressible Flows

Computational fluid dynamics (CFD) simulations of turbulent flows at large Reynolds number, e.g., $\mathrm{Re} > 10^6$, are among those problems that typically require a huge amount of computational resources in order to resolve the turbulent flow structures in space and time, and have been addressed as an application by the ExaDG project. The underlying model problem is given by the incompressible Navier–Stokes equations

$$\frac{\partial u}{\partial t} + \nabla \cdot (u \otimes u) - \nu \nabla^2 u + \nabla p = f \ , \tag{20}$$

$$\nabla \cdot u = 0 \ . \tag{21}$$

Scale-resolving simulations for engineering applications typically involve beyond $O(10^{10} - 10^{11})$ unknowns (DoFs) and $O(10^5 - 10^7)$ time steps. High-performance implementations for this type of problem are therefore of paramount importance for the CFD community. It is important to stress that implementing a given algorithm optimally for a given hardware, i.e., an implementation that performs close to the hardware limits, is only one step to achieve the goal of providing efficient flow solvers for engineering problems as emphasized in the introduction. While the previous sections discussed the second and third term in Eq. (1), namely the performance of matrix-free evaluation routines and fast multigrid solvers for high-order discretizations, we now also include discretization aspects into the discussion. The implementation makes use of the fast matrix-free evaluation routines and multigrid solvers discussed in previous sections.

We use a method of lines approach with high-order DG discretizations in space and splitting methods with BDF time integration. Splitting methods separate the solution of the incompressible Navier–Stokes equations into sub-problems such as a Poisson equation for the pressure and a (convection–)diffusion equation for the velocity and are among the most efficient solvers currently known. In a first contribution [45], we highlighted that previous discretization methods lack robustness, on the one hand in the limit of small time step sizes, and on the other

hand in under-resolved scenarios where the spatial discretization only resolves the largest scales of the flow. The stability problem for small time step sizes has been addressed in detail in [21] where we found that a proper DG discretization of velocity-pressure coupling terms is essential to achieve robustness at small time steps. In [23], we presented the first high-order DG incompressible flow solver that is robust in the under-resolved regime and that relies completely on efficient matrix-free evaluation routines. The developed discretization approach is attractive as it provides a generic solver for turbulent flow simulations that is robust and accurate without the use of explicit turbulence models. Such a technique is known as implicit large-eddy simulation in the literature and has the advantage that it does not require turbulence model parameters. While this property of high-order DG discretizations is already known from discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, the work [23] has been the first demonstrating this property for DG discretizations of the incompressible Navier–Stokes equations. The key ingredient for a robust high-order, L_2-conforming DG discretization for incompressible flows turns out to be the use of consistent stabilization terms that enforce the divergence-free constraint and inter-element mass conservation in a weak sense. These requirements can also be included into the finite element function spaces by using so-called $H(\text{div})$-conforming (normal-continuous) discretizations that are exactly (pointwise) divergence-free by using Raviart–Thomas elements. As investigated in detail in [26], such an approach has indeed very similar discretization properties when compared with the stabilized L_2-conforming approach in practically relevant, under-resolved application scenarios. The model has been extended to moving meshes in [20].

A detailed performance analysis has been undertaken in [22] where we discuss the incompressible flow solver w.r.t. its efficiency according to Eq. (1). Based on this efficiency model, we have then compared matrix-free solvers based on incompressible and compressible Navier–Stokes formulations in [24] for under-resolved turbulent incompressible flows. The compressible solver uses explicit time integration and therefore only requires one operator evaluation in every Runge–Kutta stage as opposed to the incompressible solver involving the solution of linear system of equations such as a pressure Poisson equation within every time step. Simple explicit solvers are often considered efficient due to better parallel scalability since implicit Krylov solvers with multigrid preconditioning involve global communication. However, our work shows a significant performance advantage of the incompressible formulation over the compressible one on the node-level for sufficient workload. Albeit speed-up factors are higher, it is difficult to achieve a performance advantage for the algorithmically simple, explicit-in-time compressible solver in the strong-scaling limit in terms of absolute run time. In our experience, the potential to outperform an implicit solver at some point in the strong-scaling limit has not materialized. We see it as a future challenge to devise optimal PDE solvers providing good performance over a wide range of problems and hardware platforms due to this high degree of interdisciplinarity.

We have applied this solver framework to conduct direct numerical simulations of turbulent channel flow in [45], the first direct numerical simulation of the turbulent

www.dbooks.org

flow over a periodic hill at Re $\approx 10^4$ in [46], and to large-eddy simulation of the FDA benchmark nozzle problem in [25]. Furthermore, we have developed multiscale wall modeling approaches that allow to use the proposed highly efficient schemes also for industrial cases with even higher Reynolds numbers than what is feasible for wall-resolved large eddy simulation [47].

Here, we show performance results obtained on SuperMUC-NG with Intel Skylake CPUs. We study the three-dimensional Taylor–Green vortex problem as a standard benchmark to assess the accuracy and computational efficiency of incompressible turbulent flow solvers. Regarding discretization accuracy and from a physical point of view, the quantity of interest is the kinetic energy dissipation rate shown in Fig. 12 as a function of time $0 \leq t \leq T = 20$ for increasing Reynolds numbers Re $= 100, 200, 400, 800, 1600, 3000, 10{,}000, \infty$. The first direct numerical simulation for the Re $= 1600$ case with a high-order DG scheme of the incompressible Navier–Stokes equations with a resolution of 1024^3 and polynomial degrees $k = 3, 7$ has been shown in [22]. Here, we show results for effective resolutions up to 3072^3 (corresponding to $0.99 \cdot 10^{11}$ DoFs) for the highest Reynolds number cases. Despite these fine resolutions, grid-converged results are achieved only up to Re $= 3000$. The inviscid problem (Re $= \infty$) is most challenging, and the results in Fig. 12 suggest that even finer resolutions are required for grid-convergence, a goal that might be achievable in the foreseeable future. The largest problem with $0.99 \cdot 10^{11}$ DoFs involved $6.6 \cdot 10^4$ time steps and required 11.4 h of wall time on 152,064 cores. In terms of degrees of freedom solved per time step per core, this results in a throughput of 1.05 MDoFs/s/core.

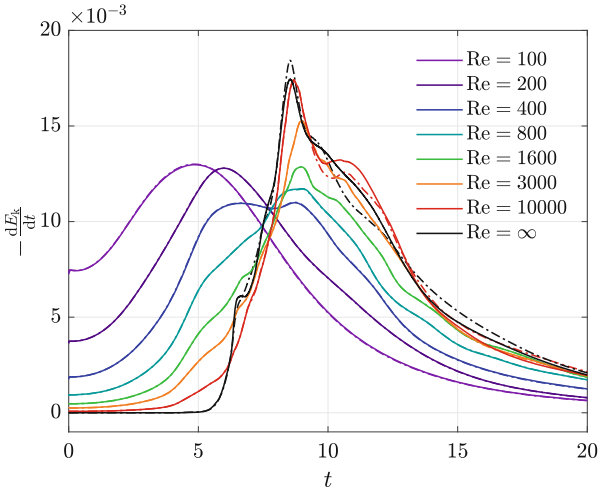


Fig. 12 Taylor–Green vortex: Kinetic energy dissipation rates for two different problem sizes (fine mesh as solid line and coarse mesh as dashed-dotted line) for each Re number: The polynomial degree is $k = 3$ and the effective resolutions $N_{\text{eff}} = (N_{\text{ele,1d}}(k+1))^3$ considered are $N_{\text{eff}} = 64^3, 128^3$ for Re $= 100$, $N_{\text{eff}} = 128^3, 256^3$ for Re $= 200, 400$, $N_{\text{eff}} = 256^3, 512^3$ for Re $= 800$, $N_{\text{eff}} = 1024^3, 2048^3$ for Re $= 1600$, and $N_{\text{eff}} = 2048^3, 3072^3$ for Re $= 3000, 10000, \infty$

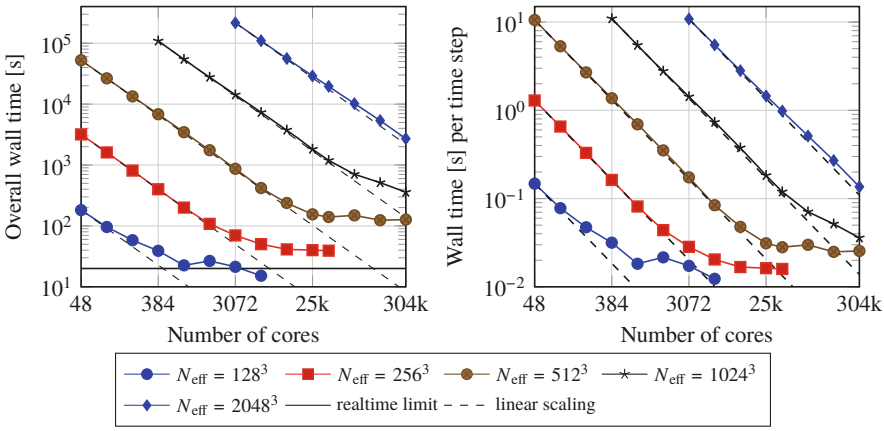


Fig. 13 Scaling analysis for incompressible flow solver on 3D Taylor–Green vortex with polynomial degree $k = 3$ at $\text{Re} = 1600$ and spatial resolutions of 128^3, 256^3, 512^3, 1024^3, 2048^3

Figure 13 shows strong scaling results for the TGV problem at $\text{Re} = 1600$ for effective resolutions of 128^3, 256^3, 1024^3, 2048^3 and polynomial degree $k = 3$. We assess strong scalability in terms of absolute run times for the whole application (including mesh-generation, setup of data structures, solvers, preconditioners, and postprocessing) rather than normalized speed-up factors as the aim of strong scalability is not only reducing but also minimizing time-to-solution, i.e., demonstrating strong-scalability of a code with poor serial performance is meaningless. The results in Fig. 13 reveal that we are able to perform the TGV simulations in realtime ($t_{\text{wall}} \leq T = 20\text{s}$) for spatial resolutions up to 128^3. These numbers can be considered outstanding and we are not aware of other high-order DG solvers achieving this performance, see also the discussions in [22, 24]. The minimum wall time in the strong-scaling limit increases on finer meshes due to more time steps (the time step size is restricted according to the CFL condition, $\Delta t \sim 1/h$, for the mixed explicit–implicit splitting solver used here). For this reason, we also show strong scalability in terms of the wall time per time step, to allow extrapolations of how many time steps can be solved within a given wall time limit which is the typical use case for large-eddy and direct numerical simulations of turbulent flows. In this metric, the curves level off around $0.02 - 0.03$ s of wall time per time step, independently of the spatial resolution. The SuperMUC-NG machine with $3 \cdot 10^5$ cores is too small to show the strong scaling limit for the largest problem size with 2048^3 resolution considered here. A parallel efficiency of 80.6% is achieved with a speed-up factor of 79.8 when scaling from 3072 cores to 304,128 cores.

www.dbooks.org

7 hyper.deal: Extending the Matrix-Free Kernels to Higher Dimensions

The matrix-free kernels developed within the ExaDG project have been implemented in a recursive manner which enables compilation with arbitrary spatial dimension. In order to be compatible with the mesh infrastructure of deal.II which is restricted to dimensions up to 3, we have developed schemes working on a tensor product of two deal.II meshes. This allows extension to 2+2, 2+3, and 3+3 dimensions. The corresponding framework is currently under development as the deal.II-extension `hyper.deal` [59].

The major application that we have in mind are kinetic problems in phase space where we use the tensor product of a spatial and a velocity mesh. However, other applications might arise such as parameter-dependent flow problems. Table 6 gives an overview of computational times on a six-dimensional Vlasov–Poisson problem, which involves an advection in the 6D space of the particle density in x and v space and the solution of a 3D Poisson equation for finding the electric potential that in turn specifies the electric field that transports the density field (cf. [44] for the same application tackled with a semi-Lagrangian solver).

Figure 14 lists the throughput of the matrix-free evaluation of cell integrals for the multi-dimensional advection in three to six spatial dimensions for polynomial degrees $k = 2, 3, 4, 5$ for AVX2 and AVX-512 vectorization over elements, respectively, without any application-specific tuning at this stage. While throughput is very good in 3D and 4D as well as 5D up to $k = 4$, performance drops significantly in 6D because the local arrays in sum factorization exhaust caches,

Table 6 Contributions to run time on 6D Vlasov–Poisson system on 320 cores with 8.6 billion spatial DoFs over 42 time steps

Category	6D advect total (of which MPI exchange)	integrate v	3D Poisson + electric field
time [s]	560 (130)	13.0	35.9

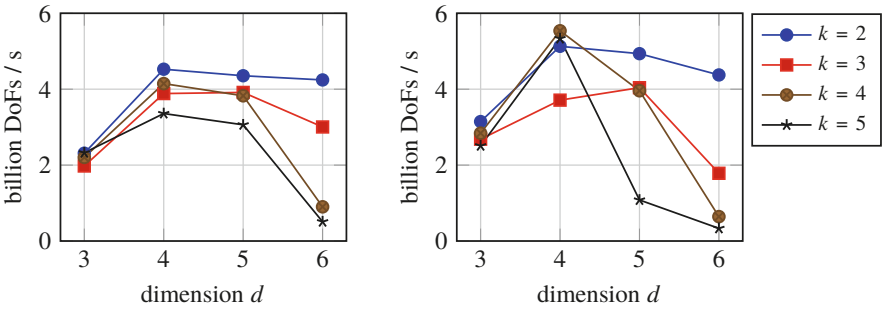


Fig. 14 Throughput of cell term for advection as a function of the spatial dimension on Intel Skylake with 4-wide vectorization (left) and 8-wide vectorization (right)

especially with AVX-512. Vectorization strategies within an element [50] are currently under development.

8 Outlook

Our work in the ExaDG project presented in this text has resulted in a highly competitive finite element framework. We have demonstrated excellent performance both for the pure operator evaluation, demonstrated e.g. by the CEED benchmark problems, as well as on an application level in computational fluid dynamics. We plan to engage in benchmarking also in the future to establish best-practices for the high-order finite element community. Furthermore, the evolving hardware landscape requires a continued effort, with increasing pressure to additional performance improvements on throughput architectures such as GPUs and FPGAs. In addition, we plan to extend our hybrid hp-multigrid framework to also handle hp-adaptive meshes. Finally, while the results from the Schwarz-based multigrid smoothers are very promising from a mathematical point of view, further steps are necessary to make them perform optimally on massively parallel hardware, and it is not yet clear how an optimal implementation compares in time-to-solution against the simpler Chebyshev-based ingredients we have considered on the large scale so far.

References

1. Alzetta, G., Arndt, D., Bangerth, W., Boddu, V., Brands, B., Davydov, D., Gassmoeller, R., Heister, T., Heltai, L., Kormann, K., Kronbichler, M., Maier, M., Pelteret, J.P., Turcksin, B., Wells, D.: The `deal.II` library, version 9.0. J. Numer. Math. **26**(4), 173–184 (2018). https://doi.org/10.1515/jnma-2018-0054
2. Anderson, R., Barker, A., Bramwell, J., Cerveny, J., Dahm, J., Dobrev, V., Dudouit, Y., Fisher, A., Kolev, T., Stowell, M., Tomov, V.: MFEM: modular finite element methods (2019). mfem.org
3. Antonietti, P.F., Sarti, M., Verani, M., Zikatanov, L.T.: A uniform additive Schwarz preconditioner for high-order discontinuous Galerkin approximations of elliptic problems. J. Sci. Comput. **70**(2), 608–630 (2017). https://doi.org/10.1007/s10915-016-0259-9
4. Arndt, D., Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kronbichler, M., Maier, M., Pelteret, J.-P., Turcksin, B., Wells, D.: The deal.II finite element library: Design, features, and insights. Comput. Math. Appl. (2020). https://doi.org/10.1016/j.camwa.2020.02.022
5. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Turek, S.: Hardware-based efficiency advances in the EXA-DUNE project. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale computing—SPPEXA 2013-2015, pp. 3–23. Springer, Cham (2016)
6. Bastian, P., Müller, E.H., Müthing, S., Piatkowski, M.: Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. J. Comput. Phys. **394**, 417–439 (2019). https://doi.org/10.1016/j.jcp.2019.06.001
7. Bauer, S., Drzisga, D., Mohr, M., Rüde, U., Waluga, C., Wohlmuth, B.: A stencil scaling approach for accelerating matrix-free finite element implementations. SIAM J. Sci. Comput. **40**(6), C748–C778 (2018)

www.dbooks.org

8. Bergen, B., Hülsemann, F., Rüde, U.: Is 1.7×10^{10} unknowns the largest finite element system that can be solved today? In: Proceeding of ACM/IEEE Conference Supercomputing (SC'05), pp. 5:1–5:14 (2005). https://doi.org/10.1109/SC.2005.38
9. Brandt, A.: Multi-level adaptive solutions to boundary-value problems. Math. Comput. **31**, 333–390 (1977). https://doi.org/10.1090/S0025-5718-1977-0431719-X
10. Brenner, S.C.: Korn's inequalities for piecewise H^1 vector fields. Math. Comput. **73**(247), 1067–1087 (2004)
11. Brown, J.: Efficient nonlinear solvers for nodal high-order finite elements in 3D. J. Sci. Comput. **45**(1–3), 48–63 (2010)
12. Cantwell, C.D., Sherwin, S.J., Kirby, R.M., Kelly, P.H.J.: Form h to p efficiently: Selecting the optimal spectral/hp discretisation in three dimensions. Math. Model. Nat. Phenom. **6**, 84–96 (2011)
13. Cantwell, C.D., Moxey, D., Comerford, A., Bolis, A., Rocco, G., Mengaldo, G., De Grazia, D., Yakovlev, S., Lombard, J.E., Ekelschot, D., Jordi, B., Xu, H., Mohamied, Y., Eskilsson, C., Nelson, B., Vos, P., Biotto, C., Kirby, R.M., Sherwin, S.J.: Nektar++: An open-source spectral/hp element framework. Comput. Phys. Comm. **192**, 205–219 (2015). https://doi.org/10.1016/j.cpc.2015.02.008
14. Charrier, D.E., Hazelwood, B., Tutlyaeva, E., Bader, M., Dumbser, M., Kudryavtsev, A., Moskovsky, A., Weinzierl, T.: Studies on the energy and deep memory behaviour of a cache-oblivious, task-based hyperbolic PDE solver. Int. J. High Perf. Comput. Appl. **33**(5), 973–986 (2019). https://doi.org/10.1177/1094342019842645
15. Clevenger, T.C., Heister, T., Kanschat, G., Kronbichler, M.: A flexible, parallel, adaptive geometric multigrid method for FEM. Technical report, arXiv:1904.03317 (2019)
16. Davydov, D., Kronbichler, M.: Algorithms and data structures for matrix-free finite element operators with MPI-parallel sparse multi-vectors. ACM Trans. Parallel Comput. (2020). https://doi.org/10.1145/3399736
17. Davydov, D., Heister, T., Kronbichler, M., Steinmann, P.: Matrix-free locally adaptive finite element solution of density-functional theory with nonorthogonal orbitals and multigrid preconditioning. Phys. Status Solidi B: Basic Solid State Phys. **255**(9), 1800069 (2018). https://doi.org/10.1002/pssb.201800069
18. Davydov, D., Pelteret, J.P., Arndt, D., Kronbichler, M., Steinmann, P.: A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid. Int. J. Numer. Meth. Eng. (2020). https://doi.org/10.1002/nme.6336
19. Deville, M.O., Fischer, P.F., Mund, E.H.: High-order Methods for Incompressible Fluid Flow, vol. 9. Cambridge University, Cambridge (2002)
20. Fehn, N., Heinz, J., Wall, W.A., Kronbichler, M.: High-order arbitrary Lagrangian-Eulerian discontinuous Galerkin methods for the incompressible Navier-Stokes equations. Technical report, arXiv:2003.07166 (2020).
21. Fehn, N., Wall, W.A., Kronbichler, M.: On the stability of projection methods for the incompressible Navier–Stokes equations based on high-order discontinuous Galerkin discretizations. J. Comput. Phys. **351**, 392–421 (2017). https://doi.org/10.1016/j.jcp.2017.09.031
22. Fehn, N., Wall, W.A., Kronbichler, M.: Efficiency of high-performance discontinuous Galerkin spectral element methods for under-resolved turbulent incompressible flows. Int. J. Numer. Meth. Fluids **88**(1), 32–54 (2018). https://doi.org/10.1002/fld.4511
23. Fehn, N., Wall, W.A., Kronbichler, M.: Robust and efficient discontinuous Galerkin methods for under-resolved turbulent incompressible flows. J. Comput. Phys. **372**, 667–693 (2018). https://doi.org/10.1016/j.jcp.2018.06.037
24. Fehn, N., Wall, W.A., Kronbichler, M.: A matrix-free high-order discontinuous Galerkin compressible Navier–Stokes solver: a performance comparison of compressible and incompressible formulations for turbulent incompressible flows. Int. J. Numer. Meth. Fluids **89**(3), 71–102 (2019). https://doi.org/10.1002/fld.4683

25. Fehn, N., Wall, W.A., Kronbichler, M.: Modern discontinuous Galerkin methods for the simulation of transitional and turbulent flows in biomedical engineering: a comprehensive LES study of the FDA benchmark nozzle model. Int. J. Numer. Meth. Biomed. Eng. **35**(12), e3228 (2019). https://doi.org/10.1002/cnm.3228
26. Fehn, N., Kronbichler, M., Lehrenfeld, C., Lube, G., Schroeder, P.W.: High-order DG solvers for under-resolved turbulent incompressible flows: a comparison of L^2 and $H(\mathrm{div})$ methods. Int. J. Numer. Meth. Fluids **91**(11), 533–556 (2019). https://doi.org/10.1002/fld.4763
27. Fehn, N., Munch, P., Wall, W.A., Kronbichler, M.: Hybrid multigrid methods for high-order discontinuous Galerkin discretizations. J. Comput. Phys. (2020). https://doi.org/10.1016/j.jcp.2020.109538
28. Fischer, P., Kerkemeier, S., Peplinski, A., Shaver, D., Tomboulides, A., Min, M., Obabko, A., Merzari, E.: Nek5000 Web page (2019). https://nek5000.mcs.anl.gov
29. Fischer, P., Min, M., Rathnayake, T., Dutta, S., Kolev, T., Dobrev, V., Camier, J.S., Kronbichler, M., Warburton, T., Świrydowicz, K., Brown, J.: Scalability of high-performance PDE solvers. Int. J. High Perf. Comput. Appl. (2020). https://doi.org/10.1177/1094342020915762
30. Gholami, A., Malhotra, D., Sundar, H., Biros, G.: FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube. SIAM J. Sci. Comput. **38**(3), C280–C306 (2016). https://doi.org/10.1137/15M1010798
31. Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., Wohlmuth, B.: Towards textbook efficiency for parallel multigrid. Numer. Math.-Theory Me. Appl. **8**(1), 22–46 (2015)
32. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. J. Comput. Sci. **17**, 509–521 (2016). https://doi.org/10.1016/j.jocs.2016.06.006. http://www.sciencedirect.com/science/article/pii/S1877750316301077. Recent Advances in Parallel Techniques for Scientific Computing
33. Hager, G., Wellein, G.: Introduction to High Performance Computing for Scientists and Engineers. CRC Press, Boca Raton (2011)
34. Hansbo, P., Larson, M.G.: Discontinuous Galerkin methods for incompressible and nearly incompressible elasticity by Nitsche's method. Comput. Methods Appl. Mech. Eng. **191**, 1895–1908 (2002)
35. Ibeid, H., Olson, L., Gropp, W.: FFT, FMM, and multigrid on the road to exascale: performance challenges and opportunities. J. Parallel Distrib. Comput. **136**, 63–74 (2020). https://doi.org/10.1016/j.jpdc.2019.09.014
36. Janssen, B., Kanschat, G.: Adaptive multilevel methods with local smoothing for H^1- and H^{curl}-conforming high order finite element methods. SIAM J. Sci. Comput. **33**(4), 2095–2114 (2011). https://doi.org/10.1137/090778523
37. Kanschat, G.: Multi-level methods for discontinuous Galerkin FEM on locally refined meshes. Comput. Struct. **82**(28), 2437–2445 (2004). https://doi.org/10.1016/j.compstruc.2004.04.015
38. Kanschat, G.: Robust smoothers for high order discontinuous Galerkin discretizations of advection-diffusion problems. J. Comput. Appl. Math. **218**, 53–60 (2008). https://doi.org/10.1016/j.cam.2007.04.032
39. Kanschat, G., Mao, Y.: Multigrid methods for $\mathbf{H}^{\mathrm{div}}$-conforming discontinuous Galerkin methods for the Stokes equations. J. Numer. Math. **23**(1), 51–66 (2015). https://doi.org/10.1515/jnma-2015-0005
40. Kempf, D., Hess, R., Müthing, S., Bastian, P.: Automatic code generation for high-performance discontinuous Galerkin methods on modern architectures. Technical report, arXiv:1812.08075 (2018)
41. Knepley, M.G., Brown, J., Rupp, K., Smith, B.F.: Achieving high performance with unified residual evaluation. Technical report, arXiv:1309.1204 (2013)
42. Kormann, K.: A time-space adaptive method for the Schrödinger equation. Commun. Comput. Phys. **20**(1), 60–85 (2016). https://doi.org/10.4208/cicp.101214.021015a

www.dbooks.org

43. Kormann, K., Kronbichler, M.: Parallel finite element operator application: graph partitioning and coloring. In: Proceeding of 7th IEEE International Conference eScience, pp. 332–339 (2011). https://10.1109/eScience.2011.53
44. Kormann, K., Reuter, K., Rampp, M.: A massively parallel semi-Lagrangian solver for the six-dimensional Vlasov–Poisson equation. Int. J. High Perform. Comput. Appl. **33**(5), 924–947 (2019). https://doi.org/10.1177/1094342019834644
45. Krank, B., Fehn, N., Wall, W.A., Kronbichler, M.: A high-order semi-explicit discontinuous Galerkin solver for 3D incompressible flow with application to DNS and LES of turbulent channel flow. J. Comput. Phys. **348**, 634–659 (2017). https://doi.org/10.1016/j.jcp.2017.07.039
46. Krank, B., Kronbichler, M., Wall, W.A.: Direct numerical simulation of flow over periodic hills up to $\mathrm{Re}_h = 10{,}595$. Flow Turbulence Combust. **101**, 521–551 (2018). https://doi.org/10.1007/s10494-018-9941-3
47. Krank, B., Kronbichler, M., Wall, W.A.: A multiscale approach to hybrid RANS/LES wall modeling within a high-order discontinuous Galerkin scheme using function enrichment. Int. J. Numer. Meth. Fluids **90**, 81–113 (2019). https://doi.org/10.1002/fld.4712
48. Kronbichler, M., Allalen, M.: Efficient high-order discontinuous Galerkin finite elements with matrix-free implementations. In: Bungartz, H.J., Kranzlmüller, D., Weinberg, V., Weismüller, J., Wohlgemuth, V. (eds.) Advances and New Trends in Environmental Informatics, pp. 89–110. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-99654-7_7
49. Kronbichler, M., Kormann, K.: A generic interface for parallel cell-based finite element operator application. Comput. Fluids **63**, 135–147 (2012). https://doi.org/10.1016/j.compfluid.2012.04.012
50. Kronbichler, M., Kormann, K.: Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM Trans. Math. Softw. **45**(3), 29:1–29:40 (2019). https://doi.org/10.1145/3325864
51. Kronbichler, M., Ljungkvist, K.: Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Trans. Parallel Comput. **6**(1), 2:1–2:32 (2019). https://doi.org/10.1145/3322813
52. Kronbichler, M., Wall, W.A.: A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. SIAM J. Sci. Comput. **40**(5), A3423–A3448 (2018). https://doi.org/10.1137/16M110455X
53. Kronbichler, M., Schoeder, S., Müller, C., Wall, W.A.: Comparison of implicit and explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. Int. J. Numer. Meth. Eng. **106**(9), 712–739 (2016). https://doi.org/10.1002/nme.5137
54. Kronbichler, M., Kormann, K., Pasichnyk, I., Allalen, M.: Fast matrix-free discontinuous Galerkin kernels on modern computer architectures. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D.E. (eds.) ISC High Performance 2017, LNCS 10266, pp. 237–255 (2017). https://doi.org/10.1007/978-3-319-58667-013
55. Kronbichler, M., Diagne, A., Holmgren, H.: A fast massively parallel two-phase flow solver for microfluidic chip simulation. Int. J. High Perf. Comput. Appl. **32**(2), 266–287 (2018). https://doi.org/10.1177/1094342016671790
56. Kronbichler, M., Kormann, K., Fehn, N., Munch, P., Witte, J.: A Hermite-like basis for faster matrix-free evaluation of interior penalty discontinuous Galerkin operators. Technical report, arXiv:1907.08492 (2019)
57. Ljungkvist, K.: Matrix-free finite-element computations on graphics processors with adaptively refined unstructured meshes. In: Proceedings of the 25th High Performance Computing Symposium, HPC '17, pp. 1:1–1:12. Society for Computer Simulation International, San Diego (2017). http://dl.acm.org/citation.cfm?id=3108096.3108097
58. Lynch, R.E., Rice, J.R., Thomas, D.H.: Direct solution of partial difference equations by tensor product methods. Numer. Math. **6**, 185–199 (1964). https://doi.org/10.1007/BF01386067
59. Munch, P., Kormann, K., Kronbichler, M.: hyper.deal: An efficient, matrix-free finite-element library for high-dimensional partial differential equations. Technical report, arXiv:2002.08110 (2020)

60. Müthing, S., Piatkowski, M., Bastian, P.: High-performance implementation of matrix-free high-order discontinuous Galerkin methods. Technical report, arXiv:1711.10885 (2017)
61. Orszag, S.A.: Spectral methods for problems in complex geometries. J. Comput. Phys. **37**, 70–92 (1980)
62. Raffenetti, K., Amer, A., Oden, L., Archer, C., Bland, W., Fujita, H., Guo, Y., Janjusic, T., Durnov, D., Blocksome, M., Si, M., Seo, S., Langer, A., Zheng, G., Takagi, M., Coffman, P., Jose, J., Sur, S., Sannikov, A., Oblomov, S., Chuvelev, M., Hatanaka, M., Zhao, X., Fischer, P., Rathnayake, T., Otten, M., Min, M., Balaji, P.: Why is MPI so slow?: Analyzing the fundamental limits in implementing MPI-3.1. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '17, pp. 62:1–62:12. ACM, New York (2017). https://doi.org/10.1145/3126908.3126963
63. Rathgeber, F., Ham, D.A., Mitchell, L., Lange, M., Luporini, F., McRae, A.T.T., Bercea, G.T., Markall, G.R., Kelly, P.H.J.: Firedrake: automating the finite element method by composing abstractions. ACM Trans. Math. Soft. **43**(3), 24:1–24:27 (2017). https://doi.org/10.1145/2998441
64. Schmidt, S.: Fast, tensor-based solution of problems involving incompressibility, Bachelor thesis. Heidelberg University, Heidelberg (2019)
65. Schoeder, S., Kormann, K., Wall, W.A., Kronbichler, M.: Efficient explicit time stepping of high order discontinuous Galerkin schemes for waves. SIAM J. Sci. Comput. **40**(6), C803–C826 (2018). https://doi.org/10.1137/18M1185399
66. Schoeder, S., Kronbichler, M., Wall, W.: Arbitrary high-order explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation. J. Sci. Comput. **76**, 969–1006 (2018). https://doi.org/10.1007/s10915-018-0649-2
67. Schoeder, S., Sticko, S., Kreiss, G., Kronbichler, M.: High-order cut discontinuous Galerkin methods with local time stepping for acoustics. Int. J. Numer. Meth. Eng. (2020). https://doi.org/10.1002/nme.6343
68. Schoeder, S., Wall, W.A., Kronbichler, M.: ExWave: A high performance discontinuous Galerkin solver for the acoustic wave equation. Soft. X **9**, 49–54 (2019). https://doi.org/10.1016/j.softx.2019.01.001
69. Solomonoff, A.: A fast algorithm for spectral differentiation. J. Comput. Phys. **98**(1), 174–177 (1992). https://doi.org/10.1016/0021-9991(92)90182-X
70. Sundar, H., Biros, G., Burstedde, C., Rudi, J., Ghattas, O., Stadler, G.: Parallel geometric-algebraic multigrid on unstructured forests of octrees. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, p. 43. IEEE Computer Society, Silver Spring (2012)
71. Świrydowicz, K., Chalmers, N., Karakus, A., Warburton, T.: Acceleration of tensor-product operations for high-order finite element methods. Int. J. High Perf. Comput. Appl. **33**(4), 735–757 (2019). https://doi.org/10.1177/1094342018816368
72. Van Loan, C.F.: The ubiquitous Kronecker product. J. Comput. Appl. Math. **123**(1–2), 85–100 (2000)
73. Van Loan, C.F., Pitsianis, N.: Approximation with Kronecker products. In: Linear Algebra for Large Scale and Real-time Applications, pp. 293–314. Springer, Berlin (1993)
74. Varga, R.S.: Matrix Iterative Analysis, 2nd edn. Springer, Berlin (2009)
75. Wichmann, K.R., Kronbichler, M., Löhner, R., Wall, W.A.: Practical applicability of optimizations and performance models to complex stencil-based loop kernels in CFD. Int. J. High Perf. Comput. Appl. **33**(4), 602–618 (2019). https://doi.org/10.1177/1094342018774126
76. Witte, J., Arndt, D., Kanschat, G.: Fast tensor product Schwarz smoothers for high-order discontinuous Galerkin methods. Technical report, arXiv:1910.11239 (2019)

www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Exa-Dune—Flexible PDE Solvers, Numerical Methods and Applications



Peter Bastian, Mirco Altenbernd, Nils-Arne Dreier, Christian Engwer, Jorrit Fahlke, René Fritze, Markus Geveler, Dominik Göddeke, Oleg Iliev, Olaf Ippisch, Jan Mohring, Steffen Müthing, Mario Ohlberger, Dirk Ribbrock, Nikolay Shegunov, and Stefan Turek

Abstract In the EXA-DUNE project we have developed, implemented and optimised numerical algorithms and software for the scalable solution of partial differential equations (PDEs) on future exascale systems exhibiting a heterogeneous massively parallel architecture. In order to cope with the increased probability of hardware failures, one aim of the project was to add flexible, application-oriented resilience capabilities into the framework. Continuous improvement of the underlying hardware-oriented numerical methods have included GPU-based sparse approximate inverses, matrix-free sum-factorisation for high-order discontinuous Galerkin discretisations as well as partially matrix-free preconditioners. On top of that, additional scalability is facilitated by exploiting massive coarse grained parallelism offered by multiscale and uncertainty quantification methods where we have focused on the adaptive choice of the coarse/fine scale and the overlap region as well as the combination of local reduced basis multiscale methods and the multilevel Monte-Carlo algorithm. Finally, some of the concepts are applied in a land-surface model including subsurface flow and surface runoff.

P. Bastian (✉) · S. Müthing
Heidelberg University, Heidelberg, Germany
e-mail: peter.bastian@iwr.uni-heidelberg.de

N.-A. Dreier · C. Engwer · J. Fahlke · R. Fritze · M. Ohlberger
University of Münster, Münster, Germany

M. Altenbernd · D. Göddeke
University of Stuttgart, Stuttgart, Germany

M. Geveler · D. Ribbrock · S. Turek
TU Dortmund, Dortmund, Germany

O. Iliev · J. Mohring · N. Shegunov
Fraunhofer ITWM, Kaiserslautern, Germany

O. Ippisch
TU Clausthal-Zellerfeld, Clausthal-Zellerfeld, Germany

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_9

www.dbooks.org

1 Introduction

In the EXA-DUNE project we extend the Distributed and Unified Numerics Environment (DUNE)[1] [6, 7] by hardware-oriented numerical methods and hardware-aware implementation techniques developed in the (now) FEAT3[2] [55] project to provide an exascale-ready software framework for the numerical solution of a large variety of partial differential equation (PDE) systems with state-of-the-art numerical methods including higher-order discretisation schemes, multi-level iterative solvers, unstructured and locally-refined meshes, multiscale methods and uncertainty quantification, while achieving close-to-peak performance and exploiting the underlying hardware.

In the first funding period we concentrated on the node-level performance as the framework and in particular its algebraic multigrid solver already show very good scalability in MPI-only mode as documented by the inclusion of DUNE's solver library in the High-Q-Club, the codes scaling to the full machine in Jülich at the time, with close to half a million cores. Improving the node-level performance in light of future exascale hardware involved multithreading ("MPI+X") and in particular exploiting SIMD parallelism (vector extensions of modern CPUs and accelerator architectures). These aspects were addressed within the finite element assembly and iterative solution phases. Matrix-free methods evaluate the discrete operator without storing a matrix, as the name implies, and promise to be able to achieve a substantial fraction of peak performance. Matrix-based approaches on the other hand are limited by memory bandwidth (at least) in the solution phase and thus typically exhibit only a small fraction of the peak (GFLOP/s) performance of a node, but decades of research have led to robust and efficient (in terms of number of iterations) iterative linear solvers for practically relevant systems. Importantly, a consideration of matrix-free and matrix-based methods needs to take the order of the method into account. For low-order methods it is imperative that a matrix entry can be recomputed in less time than it takes to read it from memory, to counteract the memory wall problem. This requires to exploit the problem structure as much as possible, i.e., to rely on constant coefficients, (locally) regular mesh structure and linear element transformations [28, 37]. In these cases it is even possible to apply stencil type techniques, like developed in the EXA-STENCIL project [40]. On the other hand, for high-order methods with tensor-product structure the complexity of matrix-free operator evaluation can be much less than that of matrix-vector multiplication, meaning that less floating-point operations have to be performed which at the same time can be executed at a higher rate due to reduced memory pressure and better suitability for vectorization [12, 39, 50]. This makes high-order methods extremely attractive for exascale machines [48, 51].

[1] http://www.dune-project.org/.

[2] http://feast.tu-dortmund.de/.

In the second funding phase we have mostly concentrated on the following aspects:

1. **Asynchronicity and fault tolerance:** High-level C++ abstractions form the basis of transparent error handling using exceptions in a parallel environment, fault-tolerant multigrid solvers as well as communication hiding Krylov methods.
2. **Hardware-aware solvers for PDEs:** We investigated matrix-based sparse-approximate inverse preconditioners including novel machine-learning approaches, vectorization through multiple right-hand sides as well as matrix-free high-order Discontinous Galerkin (DG) methods and partially matrix-free robust preconditioners based on algebraic multigrid (AMG).
3. **Multiscale (MS) and uncertainty quantification (UQ) methods:** These methods provide an additional layer of embarrassingly parallel tasks on top of the efficiently parallelized forward solvers. A challenge here is load balancing of the asynchronous tasks which has been investigated in the context of the localized reduced basis multiscale method and multilevel Monte Carlo methods.
4. **Applications:** We have considered large-scale water transport in the subsurface coupled to surface flow as an application where the discretization and solver components can be applied.

In the community, there is broad consensus on the assumptions about exascale systems that did not change much during the course of this 6 year project. A report by the Exascale Mathematics Working Group to the U.S. Department of Energy's Advanced Scientific Computing Research Program [16] summarises these challenges as follows, in line with [35] and more recently the Exascale Computing Project:[3] (1) The anticipated power envelope of 20 MW implies strong limitations on the amount and organisation of the hardware components, an even stronger necessity to fully exploit them, and eventually even power-awareness in algorithms and software. (2) The main performance difference from peta- to exascale will be through a 100–1000 fold increase in parallelism at the node level, leading to extreme levels of concurrency and increasing heterogeneity through specialised accelerator cores and wide vector instructions. (3) The amount of memory per 'core' and the memory and interconnect bandwidth/latency will only increase at a much smaller rate, hence increasing the demand for lower memory footprints and higher data locality. (4) Finally, hardware failures, and thus the mean-time-between-failure (MTBF), were expected to increase proportionally (or worse) corresponding to the increasing number of components. Recent studies have indeed confirmed this expectation [30], although not at the projected rate. First exascale systems are scheduled for 2020 in China [42], 2021 in the US and 2023 [25] in Europe. Although the details are not yet fully disclosed, it seems that the number of nodes will not be larger than 10^5 and will thus remain in the range of previous machines such as the

[3] https://www.exascaleproject.org/.

www.dbooks.org

BlueGene. The major challenge will thus be to exploit the node level performance of more than 10 TFLOP/s.

The rest of this paper is organized as follows. In Sect. 2 we lay the foundations of asynchronicity and resilience, while Sect. 3 discusses several aspects of hardware-aware and scalable iterative linear solvers. These building blocks will then be used in Sects. 4 and 5 to drive localized reduced basis and multilevel Monte-Carlo methods. Finally, Sect. 6 covers our surface-subsurface flow application.

2 Asynchronicity and Fault Tolerance

As predicted in the first funding period, latency has indeed become a major issue, both within a single node as well as between different MPI ranks. The core concept underlying all latency- and communication-hiding techniques is asynchronicity. This is also crucial to efficiently implement certain local-failure local-recovery methods. Following the DUNE philosophy, we have designed a generic layer that abstracts the use of asynchronicity in MPI from the user. In the following, we first describe this layer and its implementation, followed by representative examples on how to build middleware infrastructure on it, and on its use for s-step Krylov methods and fault tolerance beyond global checkpoint-restart techniques.

2.1 Abstract Layer for Asynchronicity

We first introduce a general abstraction for asynchronicity in parallel MPI applications, which we developed for DUNE. While we integrated these abstractions with the DUNE framework, most of the code can easily be imported into other applications, and is available as a standalone library.

The C++ API for MPI was dropped from MPI-3 since it offered no real advantage over the C bindings, beyond being a simple wrapper layer. Most MPI users coding in C++ are still using the C bindings, writing their own C++ interface/layer, in particular in more generic software frameworks. At the same time the C++11 standard introduced high-level concurrency concepts, in particular the future/promise construct to enable an asynchronous program flow while maintaining value semantics. We adopt this approach as a first principle in our MPI layer to handle asynchronous MPI operations and propose a high-level C++ MPI interface, which we provide in DUNE under the generic interface of `Dune::Communication` and a specific implementation `Dune::MPICommunication`.

An additional issue of the concrete MPI library in conjunction with C++ is the error handling concept. In C++, exceptions are the advocated approach to handle error propagation. As exceptions change the local code path on the, e.g., failing process in a hard fault scenario, exceptions can easily lead to a deadlock. As we

discuss later, the introduction of our asynchronous abstraction layer enables global error handling in an exception friendly manner.

In concurrent environments a C++ future decouples values from the actual computation (promise). The program flow can continue while a thread is computing the actual result and promotes this via promise to the future. The MPI C and Fortran interfaces offer asynchronous operations, but in contrast to thread parallel, the user does not specify the operation within the concurrent operation. Actually, MPI on its own does not offer any real concurrency at all, and provides instead a handle-based programming interface to avoid certain cases of deadlocks: the control flow is allowed to continue without finishing the communication, while the communication usually only proceeds when calls into the MPI library are executed.

We developed a C++ layer on top of the asynchronous MPI operations, which follows the design of the C++11 future. Note that the actual `std::future` class cannot be used for this purpose.

```
template<typename T>
class Future{
  void wait();
  bool ready() const;
  bool valid() const;
  T get();
};
```

As different implementations like thread-based `std::future`, task-based `TBB::future`, and our new `MPIFuture` are available, usability greatly benefits from a dynamically typed interface. This is a reasonable approach, as `std::future` is using a dynamical interface already and also the MPI operations are coarse grained, so that the additional overhead of virtual function calls is negligible. At the same time the user expects a future to offer value semantics, which contradicts the usual pointer semantics used for dynamic polymorphism. In EXA-DUNE we decided to implement type-erasure to offer a clean and still flexible user interface. An `MPIFuture` is responsible for handling all states associated with an MPI operation.

```
class MPIFuture{
private:
  mutable MPI_Request req_;
  mutable MPI_Status status_;
  impl::Buffer<R> data_;
  impl::Buffer<S> send_data_;
public:
  ...
};
```

The future holds a mutable `MPI_Request` and `MPI_Status` to access information on the current operation and it holds buffer objects, which manage the actual data. These buffers offer a great additional value, as we do not access the raw data directly, but can include data transformation and varying ownership. For example

www.dbooks.org

it is now possible to directly send an `std::vector<double>`, where the receiver automatically resizes the `std::vector` according to the incoming data stream.

This abstraction layer enables different use cases, highlighted below:

1. **Parallel C++ exception handling**: Exceptions are the recommended way to handle faults in C++ programs. As exceptions alter the execution path of a single node, they are not suitable for parallel programs. As asynchronicity allows for moderately diverging execution paths, we can use it to implement parallel error propagation using exceptions.
2. **Solvers and preconditioners tolerant to hard and soft faults**: This functionality is used for failure propagation, restoration of MPI in case of a hard fault, and asynchronous in-memory checkpointing.
3. **Asynchronous Krylov solvers**: Scalar products in Krylov methods require global communication. Asynchronicity can be used to hide the latency and improve strong scalability.
4. **Asynchronous parallel IO**: The layer allows to transform any non-blocking MPI operation into a really asynchronous operation. This allows also to support asynchronous IO, to hide the latency of write operations and overlap with the computation of the next iteration or time step.
5. **Parallel localized reduced basis methods**: Asynchronicity will be used to mitigate the load-imbalance inherent in the error estimator guided adaptive online enrichment of local reduced bases.

2.2 Parallel C++ Exception Handling

In parallel numerical algorithms, unexpected behaviour can occur quite frequently: a solver could diverge, the input of a component (e.g., the mesher) could be inappropriate for another component (e.g., the discretiser), etc. A well-written code should detect unexpected behaviour and provide users with a possibility to react appropriately in their own programs, instead of simply terminating with some error code. For C++, exceptions are the recommended method to handle this. With well placed exceptions and corresponding try-catch blocks, it is possible to accomplish a more robust program behaviour. However, the current MPI specification [44] does not define any way to propagate exceptions from one rank (process) to another. In the case of unexpected behaviour within the MPI layer itself, MPI programs simply terminate, maybe after a time-out. This is a design decision that unfortunately implies a severe disadvantage in C++, when combined with the ideally asynchronous progress of computation and communication: an exception that is thrown locally by some rank can currently lead to a communication deadlock, or ultimately even to undesired program termination. Even though exceptions are technically an illegal use of the MPI standard (a peer no longer participates in a communication), it undesirably conflicts with the C++ concept of error handling.

Building on top of the asynchronicity layer, we have developed an approach to enable parallel C++ exceptions. We follow C++11 techniques, e.g., use future-like abstractions to handle asynchronous communication. Our currently implemented

interface requires ULFM [11], an MPI extension to restore communicators after rank losses, which is scheduled for inclusion into MPI-4. We also provide a fallback solution for non-ULFM MPI installations, that employs an additional communicator for propagation and can, by construction, not handle hard faults, i.e., the loss of a node resulting in the loss of rank(s) in some communicator.

To detect exceptions in the code we have extended the `Dune::MPIGuard`, that previously only implemented the scope guard concept to detect and react on local exceptions. Our extension revokes the MPI communicator using the ULFM functionality if an exception is detected, so that it is now possible to use communication inside a block with scope guard. This makes it superfluous to call the `finalize` and `reactivate` methods of the `MPIGuard` before and after each communication.

```
try{
  MPIGuard guard(comm);
  do_something();
  communicate(comm);
}catch(...){
  comm.shrink();
  recover(comm);
}
```

Listing 1 MPIGuard

Listing 1 shows an example how to use the `MPIGuard` and recover the communicator in a node loss scenario. In this example, an exception that is thrown only on a few ranks in `do_something()` will not lead to a deadlock, since the `MPIGuard` would revoke the communicator. Details of the implementation and further descriptions are available in a previous publication [18]. We provide the "black-channel" fallback implementation as a standalone version.[4] This library uses the P-interface of the MPI standard, which makes it possible to redefine MPI functions. At the initialization of the MPI setting the library creates an opaque communicator, called blackchannel, on which a pending `MPI_Irecv` request is waiting. Once a communicator is revoked, the revoking rank sends messages to the pending blackchannel request. To avoid deadlocks, we use `MPI_Waitany` to wait for a request, which listens also for the blackchannel request. All blocking communication is redirected to non-blocking calls using the P-interface. The library is linked via `LD_PRELOAD` which makes it usable without recompilation and could be removed easily once a proper ULFM implementation is available in MPI.

Figure 1 shows a benchmark comparing the time which is used for duplicating a communicator, revoking it and restore a valid state. The benchmark was performed on PALMA2, the HPC cluster of the University of Muenster. Three implementations are compared; *OpenMPI_BC* and *IntelMPI_BC* are using the blackchannel library based on OpenMPI and IntelMPI, respectively. *OpenMPI_ULFM* uses the ULFM

[4] https://gitlab.dune-project.org/exadune/blackchannel-ulfm.

www.dbooks.org

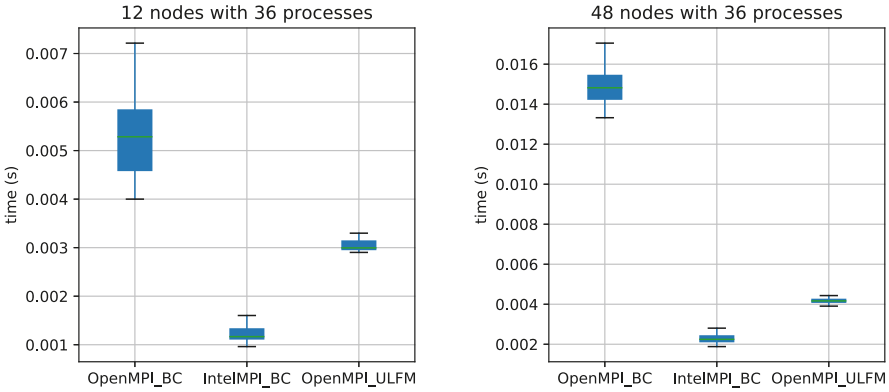


Fig. 1 Benchmark of different MPI implementations: 12 nodes with 36 processes (left), 48 nodes with 36 processes (right), cf. [18]

implementation provided by fault-tolerance.org, which is based on OpenMPI. We performed 100 measurements for each implementation. The blackchannel implementation is competitive to the ULFM implementation. As OpenMPI is in this configuration not optimized and does not use the RDMA capabilities of the interconnect, it is slower than the IntelMPI implementation. The speed up of the *OpenMPI_ULFM* version compared to the *OpenMPI_BC* version is due to the better communication strategy.

2.3 Compressed in-Memory Checkpointing for Linear Solvers

The previously described parallel exception propagation, rank loss detection and communicator restoration by using the ULFM extension, allow us to implement a flexible in-memory checkpointing technique which has the potential to recover from hard faults on-the-fly without any user interaction. Our implementation establishes a backup and recovery strategy which in part is based on a local-failure local-recovery (LFLR) [54] approach, and involves lossy compression techniques to reduce the memory footprint as well as bandwidth pressure. The contents of this subsection have not been published previously.

Modified Solver Interface To enable the use of exception propagation as illustrated in the previous section and to implement different backup recovery approaches we kept all necessary modifications to DUNE-ISTL, the linear solver library. We embed the solver initialisation and the iterative loop in a try-catch block, and provide additional entry and execution points for recovery and backup, see Listing 2 for details. Default settings are provided on the user level, i.e., DUNE-PDELAB.

```
1  init_variables();
2  done = false;
3  while (!done) try {
4    MPIGuard guard(comm);
5    if (this->processRecovery(...))
6      reinit_execution();
7    } else {
8      init_execution();
9    }
10   for (i=0 ; i<=maxit; i++ ) {
11     do_iteration();
12     if (converged) {
13       done = true;
14       break;
15     }
16     this->processBackup(...);
17   }
18 } catch(Exception & e) {
19   done = false;
20   comm.reconstitute();
21   if (!this->processOnException(...))
22     throw;
23 }
```

Listing 2 Solver modifications

This implementation ensures that the iterative solving process is active until the convergence criterion is reached. An exception inside the try-block on any rank is detected by the `MPIGuard` and propagated to all other ranks, so that all ranks will jump to the catch-block.

This catch-block can be specialised for different kind of exceptions, e.g., if a solver has diverged and a corresponding exception is thrown it could define some specific routine to define a modified restart with a possibly more robust setting and/or initial guess. The catch-block in Listing 2 exemplarily shows a possible solution in the scenario of a communicator failure, e.g., a node loss which is detected by using the ULFM extension to MPI, encapsulated by our wrapper for MPI exceptions. Following the detection and propagation, all still valid ranks end up in the catch-block and the communicator must be re-established in some way (Listing 2, line 20). This can be done by shrinking the communicator or replacing lost nodes by some previously allocated spare ones. After the communicator reconstitution a user-provided stack of functions can be executed (Listing 2, line 21) to react on the exception. If there is no on-exception-function or neither of them returns true the exception is re-thrown to the next higher level, e.g., from the linear solver to the application level, or in case of nested solvers, e.g. in optimisation or uncertainty quantification.

Furthermore, there are two additional entry points for user provided function stacks: In line 5 of Listing 2 a stack of recovery functions is executed and if it returns true, the solver expects that some modification, i.e., recovery, has been done.

www.dbooks.org

In this case it could be necessary that the other participating ranks have to update some data, like resetting their local right hand side to the initial values. The backup function stack in line 16 allows the user to provide functions for backup creation etc., after an iteration finished successfully.

Recovery Approaches First, regardless of these solver modifications, we describe the recovery concepts which are implemented into an exemplary recovery interface class providing functions that can be passed to the entry points within the modified solver. The interoperability of these components and the available backup techniques are described later. Our recovery class supports three different methods to recover from a data loss. The first approach is a global rollback to a backup, potentially involving lossy compression: progress on non-faulty ranks may be lost but the restored data originate from the same state, i.e., iteration. This means there is no asynchronous progression in the recovered iterative process but possibly just an error introduced through the used backup technique, e.g., through lossy compression. This compression error can reduce the quality of the recovery and lead to additional iterations of the solver, but is still superior to a restart, as seen later. For the second and third approaches, we follow the local-failure local-recovery strategy and re-initialize the data which are lost on the faulty rank by using a backup. The second, slightly simpler strategy uses these data to continue with solver iterations. The third method additionally smoothes out the probably deteriorated (because of compression) data by solving a local auxiliary problem [29, 31]. This problem is set up by restricting the global operator to its purely local degrees of freedom with indices $\mathcal{F} \subset \mathbb{N}$ and a Dirichlet boundary layer. The boundary layer can be obtained by extending $\mathcal{F}$ to some set $\mathcal{J}$ using the ghost layer, or possibly the connectivity pattern of the operator $\mathbf{A}$. The Dirichlet values on the boundary layer are set to their corresponding values x_N on the neighbouring ranks and thus additional communication is necessary:

$$\begin{aligned} \mathbf{A}(\mathcal{F},\mathcal{F})\tilde{x}(\mathcal{F}) &= b(\mathcal{F}) && \text{in } \mathcal{F} \\ \tilde{x} &= x_N && \text{on } \mathcal{J}\backslash\mathcal{F} \end{aligned}$$

If this problem is solved iteratively and backup data are available, the computation speed can be improved by initializing $\tilde{x}$ with the data from the backup.

Backup Techniques Our current implementation provides two different techniques for compressed backups as well as a basic class which allows 'zero'-recovery (zeroeing of lost data) if the user wants to use the auxiliary solver in case of data loss without storing any additional data during the iterative procedure.

The next backup class uses a multigrid hierarchy for lossy data compression. Thus it should only be used if a multigrid operator is already in use within the solving process because otherwise the hierarchy has to be built beforehand and introduces additional overhead. Compressing the iterative vector with the multigrid hierarchy currently involves a global communication. In addition there is no adaptive control of the compression depth (i.e., hierarchy level where the

backup is stored), but it has to be specified by the user, see a previous publication for details [29].

We also implemented a compressed backup technique based on SZ compression [41]. SZ allows compression to a specified accuracy target and can yield better compression rates than multigrid compression. The compression itself is purely local and does not involve any additional communication. We provide an SZ backup with a fixed user-specified compression target as well as a fully adaptive one which couples the compression target to the residual norm within the iterative solver. For the first we achieve an increased rate while we approach the approximate solution, as seen in Fig. 2 (top, pink lines), at the price of an increased overhead in case of a data loss (cf. Fig. 3). The backup with adaptive compression target (blue lines) gives more constant compression rates, and a better recovery in case of faults in particular in the second half of the iterative procedure of the solver.

The increased compression rate for the fixed SZ backup is obtained because, during the iterative process, the solution gets more smooth and thus can be compressed better by the algorithm. For the adaptive method this gain is counteracted by the demand of a higher compression accuracy.

All backup techniques require to communicate a data volume smaller than the volume of four full checkpoints, see Fig. 2 (bottom). Furthermore this bandwidth requirement is distributed over all 68 iterations (in the fault-free scenario) and could be decreased further by a lower checkpoint frequency.

The chosen backup technique is initiated before the recovery class and passed to it. Further backup techniques can be implemented by using the provided base class and overloading the virtual functions.

Bringing the Approaches Together The recovery class provides three functions which are added to the function stacks within the modified solver interface. The

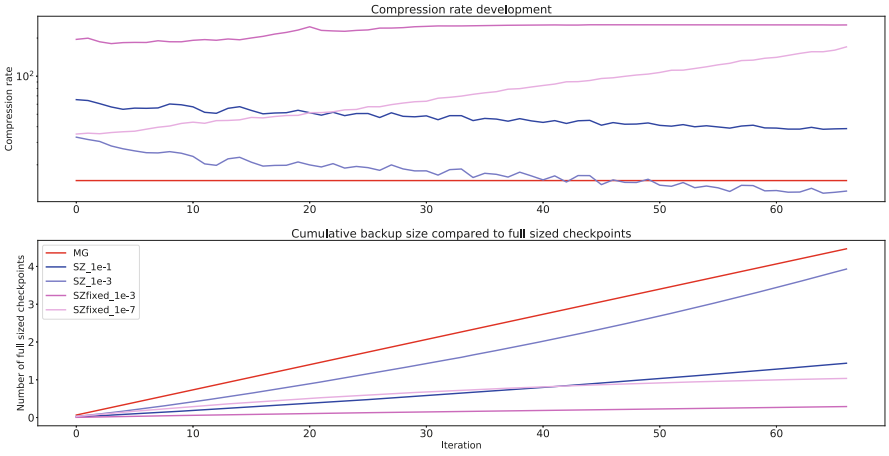


Fig. 2 Compression rate in the iterative solution for an anisotropic Poisson problem on 52 cores with approximately 480 K DOF per core

www.dbooks.org

backup routine is added to the stack of backup functions of the specified iterative solver and generates backups of the current iterative solution by using the provided backup class.

```
SomeSolver solver;
SomeBackup backup;
Recovery recovery(backup);
solver.addBackupFunction(&Recovery::backup, &recovery);
```

To adapt numerical as well as communication overhead for different fault scenarios and machine characteristics, the backup creation frequency can be varied. After the creation of the backup it is sent to a remote rank where it is kept in memory but never written to disk. In the following this is called 'remote backup'. Currently the backup propagation happens circular by rank. It is also possible to trigger writing a backup to disk.

In the near future we will implement an on-the-fly recovery if an exception is thrown. These will be provided to the other two function stacks and will differ depending on the availability of the ULFM extensions: if the extension is not available we can only detect and propagate exceptions but not recover a communicator in case of hard faults, i.e., node losses (cf. Sect. 2.2). In this scenario the function provided to the on-exception stack will only write out the global state. Fault-free nodes will write the data of the current iterative vector, whereas for faulty nodes the corresponding remote backup is written. In the following the user will be able to provide a flag to the executable which modifies the backup object initiation to read in the stored checkpoint data. Afterwards the recovery function of our interface will overwrite the initial values of the solver with the checkpointed and possibly smoothed data like described above. If the ULFM extensions are available, the recovery can be realised without any user interaction: during the backup class initiation a global communication ensures that it is the first and therefore fault-free start of the parallel execution. If the process is a respawned one which replaces a lost rank, this communication is matched by a send communication created from the rank which holds the corresponding remote backup. This communication will be initiated by the on-exception function. In addition to this message the remote backup rank sends the stored compressed backup so that the respawned rank can use this backup to recover the lost data.

So far, we have not fully implemented rebuilding the solver and preconditioner hierarchy, and the re-assembly of the local systems, in case of a node loss. This can be done with, e.g., message logging [13], or similar techniques which allow recomputing the individual data on the respawned rank without additional communication.

Figure 3 shows the effect of various combinations of different backup and recovery techniques in case of a data loss on one rank after iteration 60. The problem is an anisotropic Poisson problem with zero Dirichlet boundary conditions which reaches the convergence criterion after 68 iterations in a fault-free scenario (black

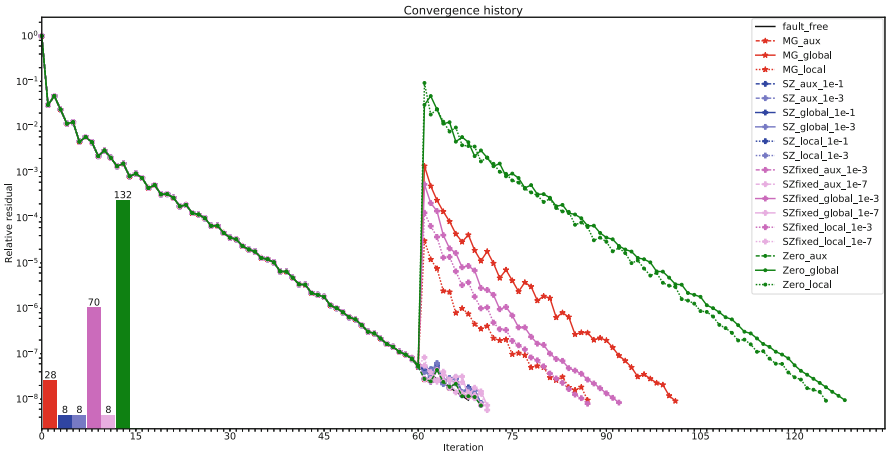


Fig. 3 Convergence history in case of data loss and recovery on one rank, same setting as in Fig. 2. Bottom left: number of iterations to solve the auxiliary problem when using the backups as initial guess. Note that the groups of the same colour are important, not the individual graphs

line). It is executed in parallel on 52 ranks with approximately 480,000 degrees of freedom per rank. Thus one rank loss corresponds to a loss of around 2% of data. For solving a conjugate gradient solver with an algebraic multigrid preconditioner is applied. In addition to the residual norm we show the number of iterations which are needed to solve the auxiliary problem when using different backups as initial guess at the bottom left.

The different backup techniques are colour-coded (multigrid: red; adaptive SZ compression: blue; fixed SZ compression: pink; no backup: green). For the SZ techniques we consider two cases, each with a different compression accuracy (fixed compression), respectively a different additional scaling coefficient (SZ). Recovery techniques are coded with different line styles: global roll-back recovery is indicated by straight lines; simple local recovery is shown with dotted lines and if an auxiliary problem is solved to improve the quality of the recovery it is drawn with a dashed line style. We observe that a zero recovery, multigrid compression and a fixed SZ backup with a low accuracy target are not competitive if no auxiliary problem is solved. The number of iterations needed until convergence then increases significantly. By applying an auxiliary solver the convergence can be almost fully restored (one additional global iteration) but the auxiliary solver needs a high amount of iterations (multigrid: 28; sz: 70; no backup: 132). Other backup techniques only need 8 auxiliary solver iterations. When using adaptive or very accurate fixed SZ compression the convergence behaviour can be nearly preserved even when only a local recovery or a global roll-back is applied. The adaptive compression technique has similar data overhead as the fixed SZ compression

www.dbooks.org

(cf. Fig. 2, bottom) but gives slightly better results: both adaptive SZ compression approaches introduce only one additional iteration for all recovery approaches. For the accurate fixed SZ compression (SZfixed_*_1e-7) we have two additional iterations when using local or global recovery but if we apply the auxiliary solver we also have only one additional iteration until convergence.

2.4 Communication Aware Krylov Solvers

In Krylov methods multiple scalar products per iteration must be computed. This involves global sums in a parallel setting. As a first improvement we merged the evaluation of the convergence criterion to the computation of a scalar product. Obviously this does not effect the computed values, but the iteration terminates one iteration later. However this reduces the number of global reductions per iteration from 3 to 2 and thus already saves communication overhead.

As a second step we modify the algorithm, such that only one global communication is performed per iteration. This algorithm can also be found in the paper of Chronopoulos and Gear [15]. Another optimization is to overlap the two scalar products with the application of the operator and preconditioner, respectively. This algorithm was first proposed by Gropp [27]. A fully elaborate version was then presented by Ghysels and Vanroose [27]. This version only needs one global reduction per iteration, which is overlapped with both the application of the preconditioner and operator. This algorithm is shown in Algorithm 2.

Algorithm 1 PCG

$r_0 = b - Ax_0$
$p_1 = Mr_0$

$\rho_1 = \langle p_1, r_0 \rangle$
for $i = 1, \ldots$ **do**
  $q_i = Ap_i$
  $\alpha_i = \langle p_i, q_i \rangle$
  $x_i = x_{i-1} + \frac{\rho_i}{\alpha_i} p_i$
  $r_i = r_{i-1} - \frac{\rho_i}{\alpha_i} q_i$
  $z_{i+1} = Mr_i$
  break if $\|r_i\| < \varepsilon$
  $\rho_{i+1} = \langle z_{i+1}, r_i \rangle$
  $p_{i+1} = \frac{\rho_{i+1}}{\rho_i} p_i + z_{i+1}$

Algorithm 2 Pipelined CG

$r_0 = b - Ax_0$
$p_1 = Mr_0$
$q_1 = Ap_1$
$\rho_1 = \langle p_1, r_0 \rangle$
$\alpha_1 = \langle p_1, q_1 \rangle$
$s_1 = Mq_1$
$t_1 = As_1$
for $i = 1, \ldots$ **do**
  $x_i = x_{i-1} + \frac{\rho_i}{\alpha_i} p_i$
  $r_i = r_{i-1} - \frac{\rho_i}{\alpha_i} q_i$
  break if $\|r_i\| < \varepsilon$
  $z_{i+1} = z_i - \frac{\rho_i}{\alpha_i} s_i$
  $w_{i+1} = w_i - \frac{\rho_i}{\alpha_i} t_i$
  $\rho_{i+1} = \langle z_{i+1}, r_i \rangle$
  $\tilde{\alpha}_{i+1} = \langle z_{i+1}, w_{i+1} \rangle$
  $\alpha_{i+1} = \frac{\alpha_i \rho_{i+1}^2}{\rho_i^2} + \tilde{\alpha}_{i+1}$
  $v_{i+1} = Mw_{i+1}$
  $u_{i+1} = Av_{i+1}$
  $s_{i+1} = \frac{\rho_{i+1}}{\rho_i} s_i + v_{i+1}$
  $t_{i+1} = \frac{\rho_{i+1}}{\rho_i} t_i + u_{i+1}$
  $p_{i+1} = \frac{\rho_{i+1}}{\rho_i} p_i + z_{i+1}$
  $q_{i+1} = \frac{\rho_{i+1}}{\rho_i} q_i + w_{i+1}$

With the new communication interface, described above, we are able to compute multiple sums in one reduction pattern and overlap the communication with computation. To apply these improvements in Krylov solvers the algorithm must be adapted, such that the communication is independent of the overlapping computation. For this adaption we extend the `ScalarProduct` interface by a function which can be passed multiple pairs of vectors for which the scalar product should be computed. The function returns a `Future` which contains a `std::vector< field_type>`, once it has finished.

```
Future<vector<field_type>>
dots(initializer_list<tuple<X&, X&>> pairs);
```

The function can be used in the Krylov methods like this:

```
scalarproduct_future = sp.dot_norm({{p,q}, {z, b}, {b,b}});
// compute while communicate
auto result = scalarproduct_future.get();
field_type p_dot_q = result[0];
field_type z_dot_b = result[1];
field_type norm_b = std::sqrt(result[2]);
```

The runtime improvement of the algorithm strongly depends on the problem size and on the hardware. On large systems the communication overhead makes up a

www.dbooks.org

Table 1 Memory requirement, computational effort and global reductions per iteration for different versions of the preconditioned conjugate gradients method

	Required memory	Additional computational effort	Global reductions
PCG	$4N$	–	2
Chronopoulos and Gear	$6N$	$1N$	1
Gropp	$6N$	$2N$	2 overlapped
Ghysels and Vanroose	$10N$	$5N$	1 overlapped

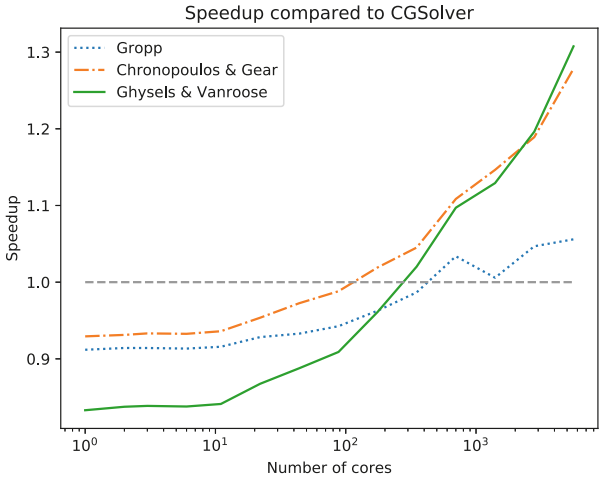


Fig. 4 Strong scaling for (pipelined) Krylov subspace methods

large part of the runtime. However, the maximum speedup is 3 for reducing the number of global reductions and 2 for overlapping communication and computation, compared to the standard version, so that a maximum speedup of 6 is possible. The optimization also increases the memory requirements and vector operations per iteration. An overview of runtime and memory requirements of the methods can be found in Table 1.

Figure 4 shows strong scaling for different methods. The shown speedup is per iteration and with respect to the `Dune::CGSolver`, which is the current CG implementation in DUNE. We use an SSOR preconditoner in an additive overlapping Schwarz setup. The problem matrix is generated from a 5-star Finite Difference model problem. With less cores the current implementation is faster than our optimized one. But with higher core count our optimized version outperforms it. The test was executed on the helics3 cluster of the University on Heidelberg, with 5600 cores on 350 nodes. We expect that on larger systems the speedup will further increase, since the communication is more expensive. The overlap of communication and computation does not really come into play, since the currently used MPI version does not support it completely.

3 Hardware-Aware, Robust and Scalable Linear Solvers

In this section we highlight improved concepts for high-performance iterative solvers. We provide matrix-based robust solvers on GPUs using sparse approximate inverses and optimize algorithm parameters using machine learning. On CPUs we significantly improve the node-level performance by using optimal matrix-free operators for Discontinous Galerkin methods, specialized partially matrix-free preconditioners as well as vectorized linear solvers.

3.1 Strong Smoothers on the GPU: Fast Approximate Inverses with Conventional and Machine Learning Approaches

In continuation of the first project phase, we enhanced the assembly of sparse approximate inverses (SPAI), a kind of preconditioner that we had shown to be very effective within the DUNE solver before [9, 26]. Concerning the assembly of such matrices we have investigated three strategies regarding their numerical efficacy (that is their quality in approximating A^{-1}), the computational complexity of the actual assembly and ultimately, the total efficiency of the amortised assembly combined with all applications during a system solution. For both strategies, this includes a decisive performance engineering for different hardware architectures with focus on the exploitation of GPUs.

SPAI-1 As a starting point we have developed, implemented and tuned a fast SPAI-1 assembly routine based on MKL/LAPACK routines (CPU) and on the cuBlas/cuSparse libraries, performing up to four times faster on the GPU. This implementation is based on the batched solution of QR decompositions that arise in Householder transformations during the SPAI minimisation process. In many cases, we observe that the resulting preconditioner features a high quality comparable to Gauss–Seidel methods. Most importantly, this result still holds true when taking into account the total time-to-solution, which includes the assembly time of the SPAI, even on a single core where the advantages of SPAI preconditioning over forward/backward substitution during the iterative solution process are not yet exploited. More systematic experiments with respect to these statements as well as their extension to larger test architectures are currently being conducted.

SAINV This preconditioner creates an approximation of the factorised inverse $A^{-1} = ZDR$ of a matrix $A \in \mathbb{R}^{N\times N}$ with D being a diagonal, Z an upper triangular and R a lower triangular Matrix.

To describe our new GPU implementation, we write the row-wise updates in the right-looking, outer product form of the A-biconjugation-process of the SAINV factorisation as follows: The assembly of the preconditioner is based on a loop over the existing rows $i \in \{1, \ldots, N\}$ of Z (initialised as unit matrix I_N), where in every iteration the loop generally calls three operations, namely a sparse-matrix vector

www.dbooks.org

Algorithm 3 Algorithm of the row-wise updates

```
for (j = i + 1, ..., N) do
  if D_jj ≠ 0 then                                   ▷ check if the fraction is unequal to zero
    α ← −(D_jj / D_ii) z_i^(i−1)
    for n = 1, ..., nnz(α) do
      if α_n > ε * max_{i,j}(A_ij) then              ▷ here α_n is the n-th entry of the vector α
        if check(z_i^n, z_j^n) then                  ▷ Has z_j already an entry at the columnindex of the n-th entry of α ?
          add(z_j^n, α_n)
          update_minimum(z_j)                        ▷ get new min. value of j-th row
        else if nnz(z_j) < ω × nnz(A)/dim(A) then    ▷ maximum number of rowentries already reached?
          insert(z_j, α_n)                           ▷ insert the value α_n at the fitting position
          update_minimum(z_j)
        else if α_n > min(z_j) then                  ▷ check if the value of α_n is bigger than the minimum of z_j
          replace(min(z_j), α_n)                     ▷ replace the old minimum with the value of α_n
          update_minimum(z_j)
```

multiplication, a dot product and an update of the remaining rows $i + 1, \ldots, N$ based on a drop-parameter ε.

In our implementation we use the ELLPACK and CSR formats, pre-allocating a fixed amount of nonzeros of the matrix Z using ω times the average number of nonzeros per row of A. Having a fixed row size, no reallocation of the arrays of the matrix format is needed and the row-wise update can be computed in parallel. This idea is based on the observation that while the density ω for typical drop tolerances is not strictly limited, it generally falls into the interval $]0, 3[$. As the SpMV and the dot kernels are well established, we take a closer look at the row-wise update, which is described more detailed in Algorithm 3. We first compute the values to be added and store them in a variable α. Then we iterate over all nonzero entries of α (which of course has the same sparsity pattern as z_i) and check if the computed value exceeds a certain drop-tolerance. If this condition is met, we have three conditions for an insertion into the matrix Z:

1. Check if there is already an existing nonzero value in the j-th row at the column index of the value α_n and search for the new minimal entry of this row.
2. Else check if there is still place in the j-th row, so we can simply insert the value α_n into that row and search for the new minimal entry of this row.
3. Else check if the value α_n is greater than the current minimum. If this condition is satisfied, then switch the old minimal value with α_n and search for the new minimal entry of this row.

If none of these conditions is met, we drop the computed value without updating the current column and repeat these steps for the next values unequal to zero of the current row. This cap of values for each row also has the following disadvantages: by having a too small maximum of nonzeros per row, a qualitative A-orthogonalization cannot be performed. To avoid this case we only take values of ω greater than one, which seems to be sufficient. Also, if a row has already reached the maximum number of nonzeros, additional but relatively small values may be dropped. This

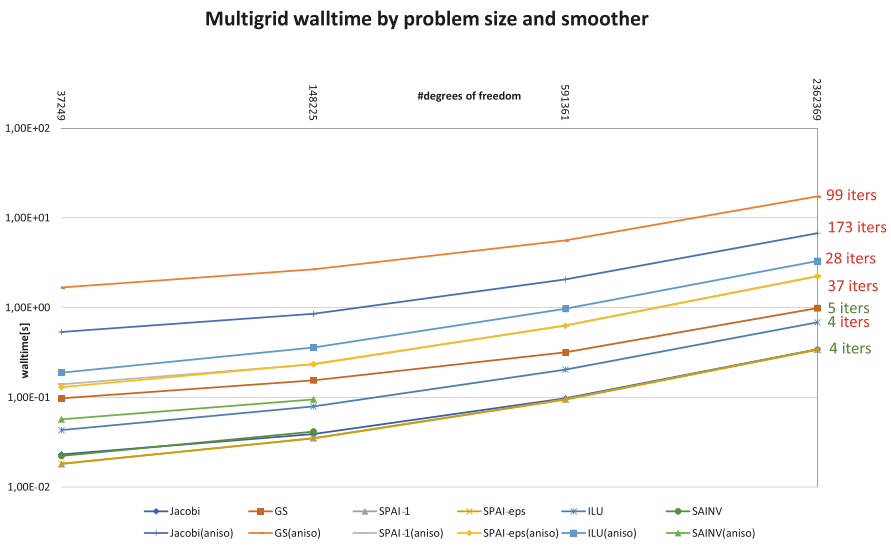


Fig. 5 GPU smoother comparison, isotropic and anisotropic Poisson benchmarks

can become an issue if the sum of these small numbers leads to a relevant entry in a later iteration. For a comparison, Fig. 5 depicts the time-to-solution for V-cycle multigrid using different strong smoothers on a P100 GPU. All smoothers are constructed using 8 Richardson iterations with (reasonably damped if necessary) preconditioners such as Jacobi, Gauss–Seidel, ILU-0, SPAI-1, SPAI-ϵ and SAINV. We set up the benchmark case from a 2D Poisson problem in the isotropic case and with two-sided anisotropies in the grid to harden the problem even for well-ordered ILU approaches. The SPAI approaches are the best choice for the smoother on the GPU.

Machine Learning Finally we started investigating how to construct approximate inverses using methods from Machine Learning [53]. The basic idea here is to treat A^{-1} as a discrete function in the course of a function regression process. The neural network therefore learns how to deduct (extrapolate) an approximation of the inverse. Once trained with many data pairs of matrices and their inverse (a sparse representation of it) a neural network like a multilayer perceptron can be able to approximate inverses rapidly. As a starting point we have employed the finite element method for the Poisson equation on different domains with linear basis functions and have used it to generate expedient systems of equations to solve. Problems of this kind are usually based on sparse M-matrices with characteristics that can be used to reduce the calculation time and effort of the neural network training and evaluation. Our results show that given the pre-defined quality of the preconditioner (equivalent to the ϵ in a SPAI-ϵ method), we can by far numerically outperform even Gauss–Seidel. Using Tensorflow [1] and numpy [4], the learning algorithm can even be performed on the GPU. Here we have used a three-layered

www.dbooks.org

fully-connected perceptron with fifty neurons in each layer plus input and output layers, and employed the resulting preconditioners in a Richardson method to solve the mentioned problem on a three times refined L-domain with a fixed number of degrees of freedom. The numerical effort of each evaluation of the neural network is basically the effort of a matrix-vector-multiplication for each layer in which the matrix size depends on the number of neurons per layer (M) and the non zero entries (N) of the input matrix, like $O(NM)$ for the first layer. The inner layers' effort, without input and output layer, just depends on the number of neurons. The crucial task now is to balance the quality of the resulting approximation and the effort to evaluate the network. We use fully connected feed-forward multilayer perceptrons as a starting point. Fully connected means that every neuron in the network is connected to each neuron of the next layer. Moreover there are no backward connections between the different layers (feed-forward). The evaluation of such neural networks is a sequence of chained matrix-vector products.

The entries of the system matrix are represented vector-wise in the input layer (cf. Fig. 6). In the same way, our output layer contains the entries of the approximate inverse. Between these layers we can add a number of hidden layers consisting of hidden neurons. How many hidden neurons we need to create strong approximate inverses is a key design decision and we discuss this below. In general our supervised training algorithm is a backward propagation with random initialisation. Alongside a linear propagation function $i_{\text{total}} = \mathbf{W} \cdot o_{\text{total}} + b$ with the total (layer) net input i_{total}, the weight matrix $\mathbf{W}$, the vector for the bias weights b and the total output of the previous layer o_{total}, we use the rectified linear unit (ReLu) function as activation function $\alpha(x)$ and thus we can calculate the output y of each neuron as $y := \alpha(\sum_{\text{j}} o_{\text{j}} \cdot w_{\text{ij}})$. Here o_{j} is the output of the preceding sending units and w_{ij} are the corresponding weights between the neurons.

For the optimization we use the L2 error function and update the weights with $w_{\text{ij}}^{(t+1)} = w_{\text{ij}}^{(t)} + \gamma \cdot o_i \cdot \delta_{\text{j}}$, with the output o_i of the sending unit and learning rate γ. δ_{j} symbolises the gradient decent method:

$$\delta_{\text{j}} = \begin{cases} f'(i_{\text{j}}) \cdot (\hat{o}_{\text{j}} - o_{\text{j}}) & \text{if neuron j is an output neuron} \\ f'(i_{\text{j}}) \cdot \sum_{k \in S} (\delta_k \cdot w_{kj}) & \text{if neuron j is a hidden neuron.} \end{cases}$$

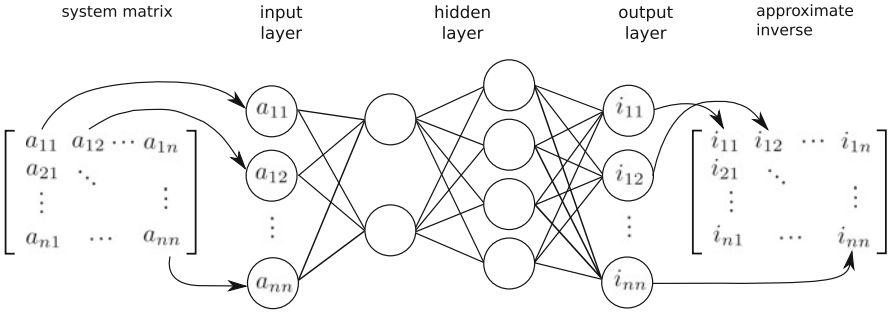


Fig. 6 Model of a neural network for matrix inversion, cf. [53]

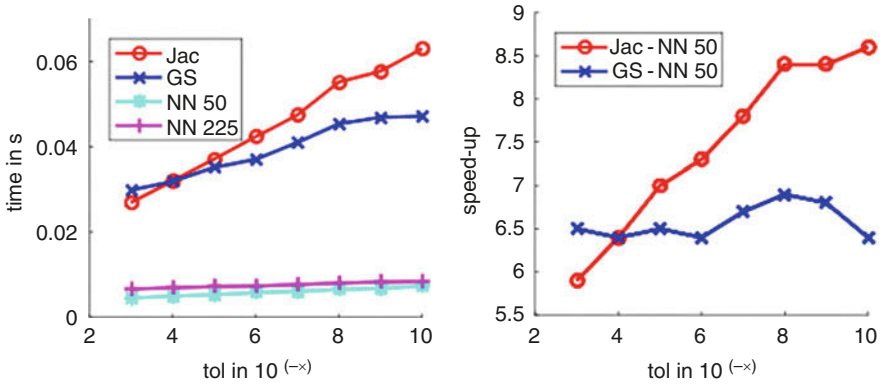


Fig. 7 Results for the defect correction with the neural network, cf. [53]

For details concerning the test/training algorithm we refer to a previous publication [53]. For the defect correction prototype, we find a significant speedup for a moderately anisotropic Poisson problem, see Fig. 7.

3.2 Autotuning with Artificial Neural Networks

Inspired by our usage of Approximate Inverses generated by artificial neural networks (ANNs), we exploit (Feed Forward-) neural networks (FNN) for the automatic tuning of solver parameters. We were able to show that it is possible to use such an approach to provide much better a-priori choices for the parametrisation of iterative linear solvers. In detailed studies for 2D Poisson problems we conducted benchmarks for many test matrices and autotuning systems using FNNs as well as convolutionary neural networks (CNNs) to predict the ω parameter in a SOR solver. In Fig. 8 we depict 100 randomly choosen samples of this study. It can be seen that even for good a-priori choices of ω the NN-driven system can compete whilst 'bad' choices (labeled constant) might lead to a stalling solver.

3.3 Further Development of Sum-Factorized Matrix-Free DG Methods

While we were able to achieve good node-level performance with our matrix-free DG methods in the first funding period, our initial implementations still did not utilize more than about 10% of the theoretical peak FLOP throughput. In the second funding period, we systematically improved on those results by focusing on several aspects of our implementation:

www.dbooks.org

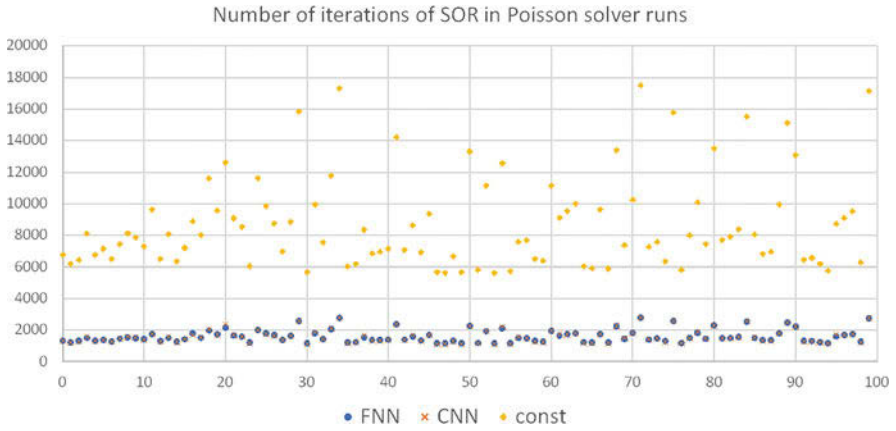


Fig. 8 Result for 100 samples of the FNN-based autotuning system for the ω parameter in SOR

Introduction of Block-Based DOF Processing Our implementation is based on DUNE-PDELAB, a very flexible discretization framework for both continuous and discontinuous discretizations of PDEs. In order to support a wide range of discretizations, PDELab has a powerful system for mapping DOFs to vector and matrix entries. Due to this flexibility, the mapping process is rather expensive. On the other hand, Discontinuous Galerkin values will always be blocked in a cell-wise manner. This can be exploited by only ever mapping the first degree of freedom associated with each cell and then assuming that all subsequent values for this cell are directly adjacent to the first entry. We have added a special 'DG codepath' to DUNE-PDELAB which implements this optimization.

Avoiding Unnecessary Memory Transfers As all of the values for each cell are stored in consecutive locations in memory, we can further optimize the framework behavior by skipping the customary gather/scatter steps before and after the assembly of each cell and facet integral. This is implemented by replacing the data buffer normally passed to the integration kernels with a dummy buffer that stores a pointer to the first entry in the global vector/matrix and directly operates on the global values. This is completely transparent to the integration kernels, as they only ever access the global data through a well-defined interface on these buffer objects. Together with the previous optimization, these two changes have allowed us to reduce the overhead of the framework infrastructure on assembly times from more than 100% to less than 5%.

Explicit Vectorization The DG implementation used in the first phase of the project was written as scalar code and relied on the compiler's auto vectorization support to utilize the SIMD instruction set of the processor, which we tried to facilitate by providing compile time loop bounds and aligned data structures. In the second phase, we have switched to explicit vectorization with a focus on AVX2, which is a common foundation instruction set across all current $\times$86-based HPC

processors. We exploit the possibilities of our C++ code base and use a well-abstracted library which wraps the underlying compiler intrinsic calls [23]. In a separate project [34], we are extending this functionality to other SIMD instruction sets like AVX512.

Loop Reordering and Fusion While vectorization is required to fully utilize modern CPU architectures, it is not sufficient. We also have to feed the execution units with a substantial number of mutually independent chains of computation (≈40–50 on current CPUs). This amount of parallelism can only be extracted from typical DG integration kernels by fusing and reordering computational loops. In contrast to other implementations of matrix-free DG assembly [22, 43], we do not group computations across multiple cells or facets, but instead across quadrature points and multiple input/output variables. In 3D, this works very well for scalar PDEs that contain both the solution itself and its gradient, which adds up to four quantities that exactly fit into an AVX2 register.

Results Table 2 compares the throughput and the hardware efficiency of our matrix-free code for two diffusion-reaction problems A (axis-parallel grid, constant coefficients per cell) and B (affine geometries, variable coefficients per cell) with a matrix-based implementation. Figure 9 compares throughput and floating point performance of our implementation for these problems as well as an additional problem C with multi-linear geometries, demonstrating that we are able to achieve more than 50% of theoretical peak FLOP rate on this machine as well as a good computational processing rate as measured in DOFs/s.

While our work in this project was mostly focused on scalar diffusion-advection-reaction problems, we have also applied the techniques shown here to projection-based Navier–Stokes solvers [51]. One important lesson learned was the unsustainable amount of work required to extend our approach to different problems and/or hardware architectures. This led us to develop a Python-based code generator in a new project [34], which provides powerful abstractions for the building blocks listed above. This toolbox can be extended and combined in new ways to achieve performance comparable to hand-optimized code. Especially for more complex problems involving systems of equations, there are a large number of possible ways to group variables and their derivatives into sum factorization kernels due to our approach of vectorizing over multiple quantities within a single cell. The resulting search space is too large for manual exploration, which the above project solved by the addition of benchmark-driven automatic comparison of those variants. Finally, initial results show good scalability of our code as shown by the strong scaling results in Fig. 10. Our implementation shows good scalability until we reach a local problem size of just 18 cells, where we still need to improve the asynchronicity of ghost data communication and assembly.

www.dbooks.org

Table 2 Full operator application, 2× Intel Xeon E5-2698v3 2.3 GHz (32 cores), for two problems of different complexity, and for comparison matrix assembly for the simpler problem and matrix-based operator application

p	Matrix-free A		Matrix-free B		Matrix-based		Matrix assembly	
	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$	$\frac{\text{DOF}}{\text{s}}$	$\frac{\text{GFLOP}}{\text{s}}$
1	1.70×10^8	104	1.19×10^8	321	2.06×10^8	24.3	1.60×10^7	345
2	3.93×10^8	238	2.52×10^8	450	6.42×10^7	27.3	8.71×10^6	371
3	5.38×10^8	328	3.30×10^8	524	2.69×10^7	29.8	4.66×10^6	368
4	5.95×10^8	387	3.88×10^8	560	9.54×10^6	23.5	2.57×10^6	301
5	6.17×10^8	424	4.03×10^8	568	4.58×10^6	23.3	1.93×10^6	307
6	5.99×10^8	439	4.06×10^8	563	2.31×10^6	23.5	1.21×10^6	231
7	5.70×10^8	442	3.98×10^8	556	1.46×10^6	30.3	6.65×10^5	143
8	5.41×10^8	445	3.85×10^8	541	6.14×10^5	24.1	8.74×10^5	198
9	5.05×10^8	439	3.70×10^8	530	2.98×10^5	26.2	7.01×10^5	133
10	4.71×10^8	432	3.59×10^8	524	–	–	–	–

Note that the matrix-based computations use significantly smaller problem sizes due to memory constraints. p denotes the polynomial degree of the DG space

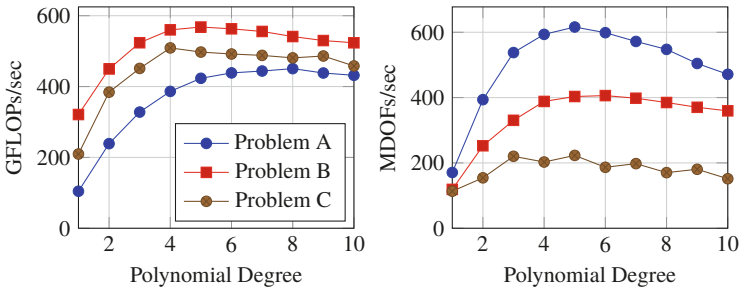


Fig. 9 Floating point performance in GFLOPs/s and throughput in MDOFs/s for full operator application, 2× Intel Xeon E5-2698v3 2.3 GHz for all model problems

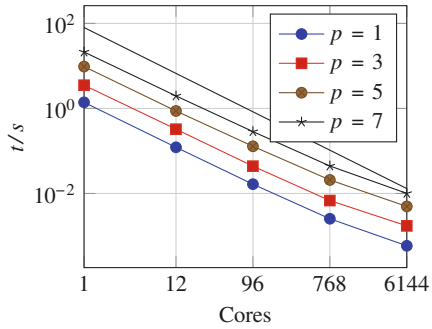


Fig. 10 Runtimes for strong scalability on IWR compute cluster (416 nodes with 2× E5-2630 v3 each, 64 GiB/node, QDR Infiniband)

3.4 Hybrid Solvers for Discontinuous Galerkin Schemes

In Sect. 3.3 we concentrated on the performance of matrix-free *operator application*. This is sufficient for instationary problems with explicit time integration, but in case of stationary problems or implicit time integration, (linear) algebraic systems need to be solved. This requires operator application and robust, scalable preconditioners.

For this we extended hybrid AMG-DG preconditioners [8] in a joint work with Eike Müller from Bath University, UK, [10]. In a solver for matrices arising from higher order DG discretizations the basic idea is to perform all computations on the DG system in a matrix-free fashion and to explicitly assemble only a matrix in a low-order subspace which is significantly smaller. In the sense of subspace correction methods [58] we employ a splitting

$$V_{DG}^p = \sum_{T \in \mathcal{T}_h} V_T^p + V_c$$

www.dbooks.org

where V_T^p is the finite element space of polynomial degree p on element T and the coarse space V_c is either the lowest-order conforming finite element space V_h^1 on the mesh $\mathcal{T}_h$, or the space of piecewise constants V_h^0. Note that the symmetric weighted interior penalty DG method from [21] reduces to the cell-centered finite volume method with two-point flux approximation on V_h^0. Note also, that the system on V_c can be assembled without assembling the large DG system.

For solving the blocks related to V_T^p, two approaches have been implemented. In the first (named *partially matrix-free*), these diagonal blocks are factorized using LAPACK and each iteration uses a backsolve. In the second approach the diagonal blocks are solved iteratively to low accuracy using matrix-free sum factorization. Both variants can be used in additive and multiplicative fashion. Figure 11 shows that the partially matrix-free variant is optimal for polynomial degree $p \leq 5$, but starting from $p = 6$, the fully matrix-free version starts to outperform all other options.

In order to demonstrate the robustness of our hybrid AMG-DG method we use the permeability field of the SPE10 benchmark problem [14] within a heterogeneous elliptic problem. This is considered to be a hard test problem in the porous media community. The DG method from [21] is employed. Figure 12 depicts results for different variants and polynomial degrees run in parallel on 20 cores. A moderate increase with the polynomial degree can be observed. With respect to time-to-solution (not reported) the additive (block Jacobi) partially matrix-free variant is to be preferred for polynomial degree larger than one.

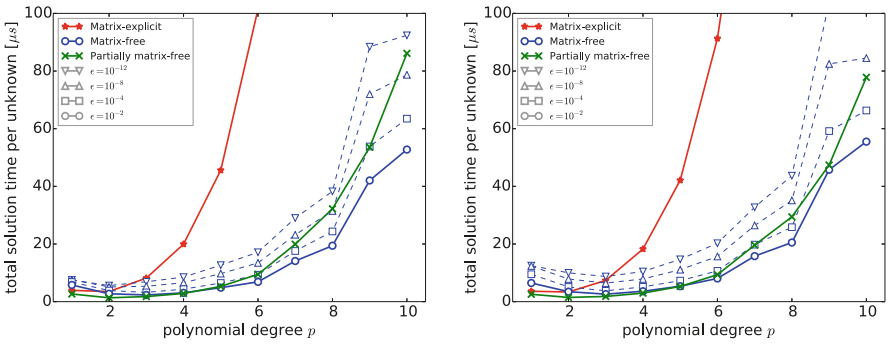


Fig. 11 Total solution time for different implementations and a range of block-solver tolerances ϵ for the Poisson problem (left) and the diffusion problem with spatially varying coefficients (right), cf. [10]

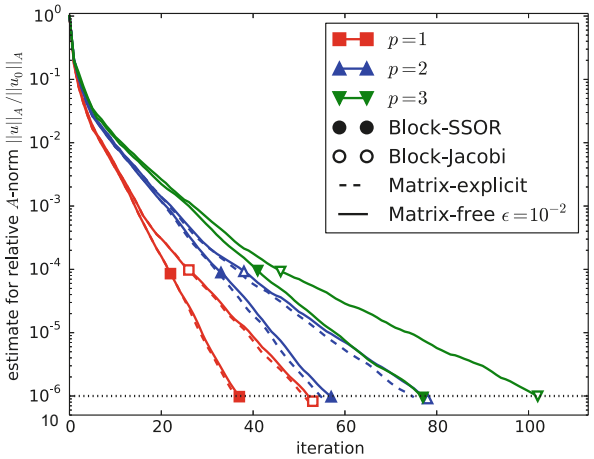


Fig. 12 Convergence history for SPE10 benchmark. The relative energy norm is shown for polynomial degrees 1 (red squares), 2 (blue upward triangles) and 3 (green downward triangles). Results for the block-SSOR smoother are marked by filled symbols and results for the block-Jacobi smoother by empty symbols. cf. [10]

3.5 *Horizontal Vectorization of Block Krylov Methods*

Methods like Multiscale FEM (see Sect. 4), optimization and inverse problems need to invert the same operator for many right-hand-side vectors. This leads to a block problem, by the following conceptual reformulation:

$$\text{foreach} \quad i \in [0, N] : \text{solve } Ax_i = b_i \qquad \rightarrow \qquad \text{solve } AX = B,$$

with matrices $X = (x_0, \ldots x_N)$, $B = (b_0, \ldots b_N)$. Such problems can be solved using Block Krylov solvers. The benefit is that the approximation space can grow faster, as the solver orthogonalizes the updates for all right-hand-sides. Even for a single right-hand-side Block Krylov based enriched Krylov methods can be used to accelerate the solution process.

Preconditioners and the actual Krylov solver can be sped up using horizontal vectorization. Assuming k right-hand-sides we observe that the scalar product yields a $k \times k$ dense matrix and has $O(k^2)$ complexity. While the mentioned larger approximation space should improve the convergence rate, this is only true for weaker preconditioners, therefore we pursued a different strategy and approximate the scalar product matrix by a sparse matrix, so that we again retain $O(k)$ complexity. In particular we consider the case of a diagonal or block-diagonal matrix. The diagonal matrix basically results in k independent solvers running in parallel, so that the performance gain is solely based on SIMD vectorization and the associated favorable memory layout.

www.dbooks.org

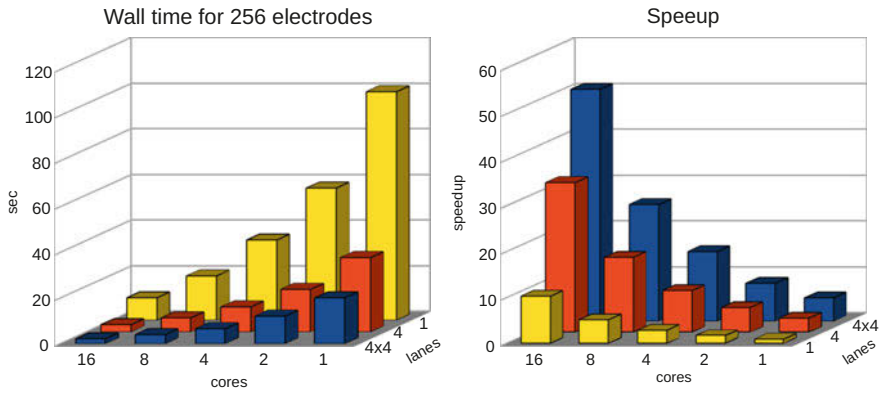


Fig. 13 Horizontal vectorization of a linear solver for 256 right-hand-side vectors. Timings on a Haswell-EP (E5-2698v3, 16 cores, AVX2, 4 lanes). Comparison with 1–16 cores and no SIMD, AVX (4 lanes), AVX (4×4 lanes)

For the implementation in DUNE-ISTL we use platform portable C++ abstractions of SIMD intrinsics, building on the VC library[38] and some DUNE specific extensions. We use this to exchange the underlying data type of the right-hand-side and the solution vector, so that we no longer store scalars, but SIMD vectors. This is possible when using generic programming techniques, like C++ templates, and yields a row-wise storage of the dense matrices X and B. This row-wise storage is optimal and ensures a high arithmetic intensity. The implementations of the Krylov solvers have to be adapted to the SIMD data types, since some operations, like casts and branches, are not available generically for SIMD data types. As a side effect, all preconditioners, including the AMG, are now fully vectorized.

Performance tests using 256 right-hand-side vectors for a 3D Poisson problem show nearly optimal speedup on a 64 core system (see Fig. 13). The tests are carried out on a Haswell-EP (E5-2698v3, 16 cores, AVX2, 4 lanes). We observe a speedup of 50, while the theoretical speedup is 64.

4 Adaptive Multiscale Methods

The main goal in the second funding phase was a distributed adaptive multilevel implementation of the localized reduced basis multi-scale method (LRBMS [49]). Like Multiscale FEM (MsFEM), LRBMS is designed to work on heterogenous multiscale or large scale problems. It performs particularly well for problems that exhibit scale separation with effects on both a fine and a coarse scale contributing to the global behavior. Unlike MsFEM, LRBMS is best applied in multi-query settings in which a parameterized PDE needs to be solved many times for different parameters. As an amalgam of domain decomposition and model order reduction

techniques, the computational domain is partitioned into a coarse grid with each macroscopic grid cell representing a subdomain for which, in an offline pre-compute stage, local reduced bases are constructed. Appropriate coupling is then applied to produce a global solution approximation from localized data. For increased approximation fidelity we can integrate localized global solution snapshots into the bases, or the local bases can adaptively be enriched in the online stage, controlled by a localized a-posteriori error estimator.

4.1 Continuous Problem and Discretization

We consider elliptic parametric multi-scale problems on a domain $\Omega \subset \mathbb{R}^d$ where we look for $p(\boldsymbol{\mu}) \in Q$ that satisfy

$$b(p(\boldsymbol{\mu}), q; \boldsymbol{\mu}) = l(q) \qquad \text{for all } q \in H_0^1(\Omega), \tag{1}$$

$\boldsymbol{\mu}$ are parameters with $\boldsymbol{\mu} \in \mathcal{P} \subset \mathbb{R}^p$, $p \in \mathbb{N}$. We let $\epsilon > 0$ be the multi-scale parameter associated with the fine scale. For demonstration purposes we consider a particular linear elliptic problem setup in $\Omega \subset \mathbb{R}^d$ $(d = 2, 3)$ that exhibits a multiplicative splitting in the quantities affected by the multi-scale parameter ϵ. It is a model for the so called global pressure $p(\boldsymbol{\mu}) \in H_0^1(\Omega)$ in two phase flow in porous media, where the total scalar mobility $\lambda(\boldsymbol{\mu})$ is parameterized. κ_ϵ denotes the heterogenous permeability tensor and f the external forces. Hence, we seek p that satisfies weakly in $H_0^1(\Omega)$,

$$-\nabla \cdot (\lambda(\boldsymbol{\mu})\kappa_\epsilon \nabla p(\boldsymbol{\mu})) = f \qquad \text{in } \Omega. \tag{2}$$

With $A(x; \boldsymbol{\mu}) := \lambda(\boldsymbol{\mu})\kappa_\epsilon(x)$ this gives rise to the following definition of the forms in (1)

$$b(p(\boldsymbol{\mu}), q; \boldsymbol{\mu}) := \int_\Omega A(\boldsymbol{\mu}) \nabla p \cdot \nabla q, \quad l(q) := \int_\Omega f q.$$

For the discretization we first require a triangulation $\mathcal{T}_H$ of Ω for the macro level. We call the elements $T \in \mathcal{T}_H$ subdomains. We then require each subdomain be covered with a fine partition $\tau_h(T)$ in a way that $\mathcal{T}_H$ and $\tau_h := \Sigma_{T \in \mathcal{T}_H} \tau_h(T)$ are nested. We denote by $\mathcal{F}_H$ the faces of the coarse triangulation and by $\mathcal{F}_h$ the faces of the fine triangulation.

Let $V(\tau_h) \subset H^2(\tau_h)$ denote any approximate subset of the broken Sobolev space $H^2(\tau_h) := \{q \in L^2(\Omega) \mid q|_t \in H^2(t) \, \forall t \in \tau_h\}$. We call $p_h(\boldsymbol{\mu}) \in V(\tau_h)$ an approximate solution of (1), if

$$b_h\big(p_h(\boldsymbol{\mu}), v; \boldsymbol{\mu}\big) = l_h(v; \boldsymbol{\mu}) \qquad \text{for all } v \in V(\tau_h). \tag{3}$$

www.dbooks.org

Here, the DG bilinear form b_h and the right hand side l_h are chosen according to the SWIPDG method [20], i.e.

$$b_h(v, w; \boldsymbol{\mu}) := \sum_{t \in \tau_h} \int_t A(\boldsymbol{\mu}) \nabla v \cdot \nabla w + \sum_{e \in \mathcal{F}(\tau_h)} b_h^e(v, w; \boldsymbol{\mu})$$

$$l_h(v; \boldsymbol{\mu}) := \sum_{t \in \tau_h} \int_t f v,$$

where the DG coupling bilinear forms b_h^e for a face e is given by

$$b_h^e(v, w; \boldsymbol{\mu}) := \int_e \langle A(\boldsymbol{\mu}) \nabla v \cdot \mathbf{n_e} \rangle [w] + \langle A(\boldsymbol{\mu}) \nabla w \cdot \mathbf{n_e} \rangle [v] + \frac{\sigma_e(\boldsymbol{\mu})}{|e|^\beta} [v][w].$$

The LRBMS method allows for a variety of discretizations, i.e. approximation spaces $V(\tau_h)$. As a particular choice of an underlying high dimensional approximation space we choose $V(\tau_h) = Q_h^k := \bigoplus_{T \in \mathcal{T}_H} Q_h^{k,T}$, where the discontinuous local spaces are defined as

$$Q_h^{k,T} := Q_h^{k,T}(\tau_h(T)) := \{q \in L^2(T) \mid q|_t \in \mathbb{P}_k(t) \, \forall t \in \tau_h(T)\}.$$

4.2 Model Reduction

For model order reduction in the LRBMS method we choose the reduced space $Q_{\text{red}} := \bigoplus_{T \in \mathcal{T}_H} Q_{\text{red}}^T \subset Q_h^k$ with local reduced approximation spaces $Q_{\text{red}}^T \subset Q_h^{k,T}$. We denote $p_{\text{red}}(\boldsymbol{\mu})$ to be the reduced solution of (3) in Q_{red}. This formulation naturally leads to solving a sparse blocked linear system similar to a DG approximation with high polynomial degree on the coarse subdomain grid.

The construction of subdomain reduced spaces Q_{red}^T is again very flexible. Initialization with shape functions on T up to order k ensures a minimum fidelity. Basis extensions can be driven by a discrete weak greedy approach which incorporates localized solutions of the global system. Depending on available computational resources, and given a suitable localizable a-posteriori error estimator $\eta(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$, we can forego computing global high-dimensional solutions altogether and only rely on online enrichment to extend Q_{red}^T 'on the fly'. With online enrichment, given a reduced solution $p_{\text{red}}(\boldsymbol{\mu})$ for some arbitrary $\boldsymbol{\mu} \in \mathcal{P}$, we first compute local error indicators $\eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$ for all $T \in \mathcal{T}_H$. If $\eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$ is greater than some prescribed bound $\delta_{\text{tol}} > 0$, we solve on a overlay region $\mathcal{N}(T) \supset T$ and extend Q_{red}^T with $p_{\mathcal{N}(T)}(\boldsymbol{\mu})|_T$. Inspired by results in [17] we set the overlay region's diameter $\text{diam}(\mathcal{N}(T))$ of the order $O(\text{diam}(T)|log(\text{diam}(T))|)$. In practice we use the completely on-line/off-line decomposable error estimator developed in [49, Sec. 4] which in turn is based on the idea of producing a

conforming reconstruction of the diffusive flux $\lambda(\boldsymbol{\mu})\kappa_\epsilon \nabla_h p_h(\boldsymbol{\mu})$ in some Raviart-Thomas-Nédélec space $V_h^l(\tau_h) \subset H_{div}(\Omega)$ presented in [21, 56].

This process is then repeated until either a maximum number of enrichment steps occur or $\eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu}) \leq \delta_{\text{tol}}$.

Algorithm 4 Schematic representation of the LRBMS pipeline

Require: $P_{\text{train}} \subset \mathcal{P}$
Require: Reconstruction operator $R_h(p_{\text{red}}(\boldsymbol{\mu})) : Q_{\text{red}}(\mathcal{T}_H) \to Q_h^k(\tau_h)$
function GREEDYBASISGENERATION(δ_{grdy}, $\eta(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$=None)
  if $\eta(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$ is not None **then**
    $E \leftarrow \{\eta(p_{\text{red}}(\mu_i), \mu_i) \mid \mu_i \in \mathcal{P}_{\text{train}}\}$
  else
    $E \leftarrow \{||R_h(p_{\text{red}}(\mu_i)) - p_h(\mu_i)|| \mid \mu_i \in \mathcal{P}_{\text{train}}\}$
  while $E \neq \emptyset$ AND $max(E) \geq \delta_{grdy}$ **do**
    $i \leftarrow \text{argmax}(E)$
    compute $p_h(\boldsymbol{\mu_i})$
    for all $T \in \mathcal{T}_H$ **do**
      extend Q_{red}^T with $p_h(\boldsymbol{\mu})|_T$
    $E \leftarrow E \setminus E_i$

Generate $\mathcal{T}_H$ ▷ Offline Phase
for all $T \in \mathcal{T}_H$ **do**
  create $\tau_h(T)$
  init Q_{red}^T with DG shape functions of order k
GREEDYBASISGENERATION($\cdots$) ▷ Optional
compute $p_{\text{red}}(\boldsymbol{\mu})$ for arbitrary$\boldsymbol{\mu}$ ▷ Online phase
for all $T \in \mathcal{T}_H$ **do** ▷ Optional Adaptive Enrichment
  $\eta \leftarrow \eta^T(p_{\text{red}}(\boldsymbol{\mu}), \boldsymbol{\mu})$
  while $\eta \geq \delta_{\text{tol}}$ **do**
    compute $p_{\mathcal{N}(T)}(\boldsymbol{\mu})$
    $Q_{\text{red}}^T \leftarrow p_{\mathcal{N}(T)}(\boldsymbol{\mu})|_T$

4.3 Implementation

We base our MPI-parallel implementation of LRBMS on the serial version developed previously. In this setup the high-dimensional quantities and all grid structures are implemented in DUNE. The model order reduction as such is implemented in Python using pyMOR [45]. The model reduction algorithms in pyMOR follow a solver agnostic design principle. Abstract interfaces allow for example projections, greedy algorithms or reduced data reconstruction to be written without knowing details of the PDE solver backend. The global macro grid $\mathcal{T}_H$ can be any MPI-enabled DUNE grid manager with adjustable overlap size for the domain decomposition, we currently use DUNE-YASPGRID. The fine grids $\tau_h(T)$ are constructed using the same grid manager as on the macro scale, with MPI subcom-

www.dbooks.org

municators.These are currently limited to a size of one (rank-local), however the overall scalability could benefit from dynamically sizing these subcommunicators to balance communication overhead and computational intensity as demonstrated in [36, Sec. 2.2]. The assembly of the local (coupling) bilinear forms is done in DUNE-GDT [24], with pyMOR/Python bindings facilitated through DUNE-XT [46], where DUNE-GRID-GLUE [19] generates necessary grid views for the SWIPDG coupling between otherwise unrelated grids. Switching to DUNE-GRID-GLUE constitutes a major step forward in robustness of the overall algorithm, compared to our previous manually implemented approach to matching independent local grids for coupling matrices assembly.

We have identified three major challenges in parallelizing all the steps in LRBMS:

1. **Global solutions $p_h(\mu)$ of the blocked system in Eq. (3) with an appropriate MPI-parallel iterative solver.** With the serial implementation already using DUNE-ISTL as the backend for matrix and vector data, we only had to generate an appropriate communication configuration for the blocked SWIPDG matrix structure to make the BiCGStab solver usable in our context. We tested this setup on the SuperMUC Petascale System in Garching. The results in Fig. 14 show very near ideal speedup from 64 nodes with 1792 MPI ranks up to a full island with 512 nodes and 14336 ranks.

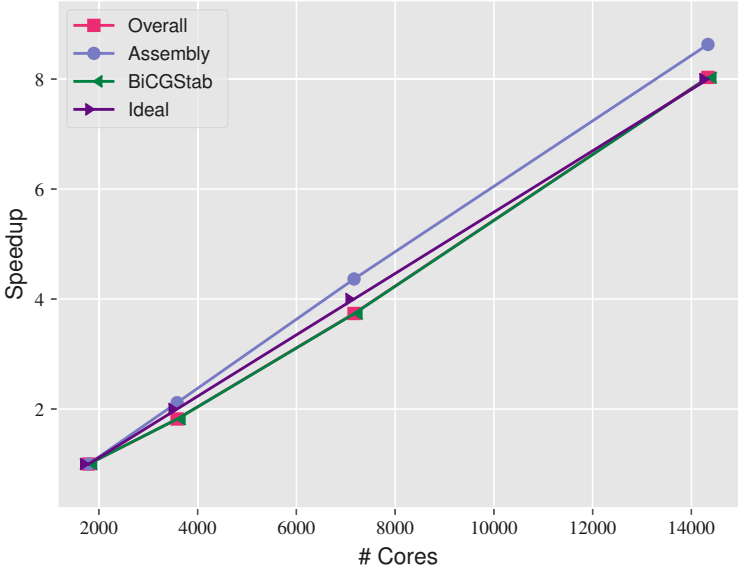


Fig. 14 Localized Reduced Basis Method: Block-SWIPDG speedup results; linear system solve (green), discretization and system assembly (blue), theoretic ideal speedup (violet) and actual achieved speedup for the overall run time (red). Simulation on $\sim 7.9 \cdot 10^6$ cubical cells shows minimum 94% parallel efficiency, scaling from 64 to 512 nodes (SuperMUC Phase 2)

2. **(Global) Reduced systems also need a distributed solver.** By design all reduced quantities in pyMOR are, at the basic, unabstracted level, NumPy arrays [57]. Therefore we cannot easily re-use the DUNE-ISTL based solvers for the high-dimensional systems. Our current implementation gathers these reduced system matrices from all MPI-ranks to rank 0, recombines them, solves the system with a direct solver and scatters the solution. There is great potential in making this step more scalable by either using a distributed sparse direct solver like Mumps [3] or translating the data into the DUNE-ISTL backend.
3. **Adaptive online enrichment is inherently load imbalanced due to its localized error estimator guidance.** The load imbalance results from one rank idling while waiting to receive updates to a basis on a subdomain in its overlap region from another rank. This idle time can be minimized by encapsulating the update in a `MPIFuture` described in Sect. 2.1. This will allow the rank to continue in its own enrichment process until the updated basis is actually needed in a subsequent step.

5 Uncertainty Quantification

The solution of stochastic partial differential equations (SPDEs) is characterized by extremely high dimensions and poses great (computational) challenges. Multilevel Monte Carlo (MLMC) algorithms attract great interest due to their superiority over the standard Monte Carlo approach. Based on Monte Carlo (MC), MLMC retains the properties of independent sampling. To overcome the slow convergence of MC, where many computationally expensive PDEs have to be solved, MLMC combines in a proper way cheap MC estimators and expensive MC estimators, achieving (much) faster convergence. One of the critical components of the MLMC algorithms is the way in which the coarser levels are selected. The exact definition of the levels is an open question and different approaches exist. In the first funding phase, Multiscale FEM was used as a coarser level in MLMC. During the second phase, the developed parallel MLMC algorithms for uncertainty quantification were further enhanced. The main focus was on exploring the capabilities of the renormalization approach for defining the coarser levels in the MLMC algorithm, and on using MLMC as a coarse grained parallelization approach.

Here, we employ MLMC to exemplarily compute the mean flux through saturated porous media with prescribed pressure drop and known distribution of the random coefficients.

Mathematical Problem As a model problem in $\mathbb{R}^2$ or $\mathbb{R}^3$, we consider steady state single phase flow in random porous media:

$$-\nabla \cdot [k(x, \omega)\nabla p(x, \omega)] = 0 \text{ for } x \in D = (0, 1)^d, \omega \in \Omega$$

www.dbooks.org

subject to the boundary conditions $p_{x=0} = 1$ and $p_{x=1} = 0$ and zero flux on other boundaries. Here p is pressure, k is scalar permeability, and ω is a random vector. The quantity of interest is the mean (expected value) $E[Q]$ of the total flux Q through the inlet of the unit cube i.e., $Q(x,\omega) := \int_{x=0} k(x,\omega)\partial_n p(x,\omega)dx$. Both the coefficient $k(x,\omega)$ and the solution $p(x,\omega)$ are subject to uncertainty, characterized by the random vector ω in a properly defined random space Ω. For generating permeability fields we consider the practical covariance $C(x,y) = \sigma^2 \exp(-||x-y||_2/\lambda)$. An algorithm based on forward and inverse Fourier transform over the circulant covariance matrix is used to generate the permeability field. For solving the deterministic PDEs a Finite Volume method on a cell centered grid is used [32]. More details and further references can be found in a previous paper [47].

Monte Carlo Simulations To quantify the uncertainty, and compute the mean of the flux we use a MLMC algorithm. Let ω_M be a random vector over a properly defined probability space, and Q_M be the corresponding flux. It is known that $E[Q_M]$ can be made arbitrarily close to $E[Q]$ by choosing M sufficiently large. The standard MC algorithm convergences very slowly, proportionally to the variance over the square root of the number of samples, which makes it often unfeasible. MLMC introduces L levels with the L-th level coinciding with the considered problem, and exploits the telescopic sum identity:

$$E[Q_M^L(\omega)] = E[Q_M^0(\omega)] + E[Q_M^1(\omega) - Q_M^0(\omega)] + \ldots E[Q_M^L(\omega) - Q_M^{L-1}(\omega)]$$

The notation $Y^l = Q^1 - Q^{l-1}$ is also used. The main idea of MLMC is to properly define levels, and combine a large number of cheap simulations, that are able to approximate the variance well, with a small number of expensive correction simulations providing the needed accuracy. For details on Monte Carlo and MLMC we refer to previous publications [32, 47] and the references therein. Here, the target is to estimate the mean flux on a fine grid, and we define the levels as discretizations on coarser grids. In order to define the permeability at the coarser levels we use the renormalization approach.

MLMC has previously run the computations at each level with the same tolerance. However, in order to evaluate the number of samples needed per level, one has to know the variance at each level. Because the exact variance is not known in advance, MLMC starts by performing simulations with a prescribed, moderate number of samples per level. The results are used to evaluate the variance at each level, and thus to evaluate the number of samples needed per level. This procedure can be repeated several times in an Estimate–Solve cycle. At each estimation step, information from all levels is needed, which leads to a synchronization point in the parallel realization of the algorithm. This may require dynamic redistribution of the resources after each new evaluation.

MLMC can provide a coarse graining in the parallelization. A well balanced algorithm has to account for several factors: (1) How many processes should be allocated per level; (2) how many processes should be allocated per deterministic problem including permeability generation; (3) how to parallelize the permeability

generation; (4) which of the parallelization algorithms for deterministic problems available in EXA-DUNE should be used; (5) should each level be parallelized separately and if not, how to group the levels for efficient parallelization. The last factor is the one giving coarse grain parallelization opportunities. For the generation of the permeability, we use the parallel MPI implementation of the FFTW library. As deterministic solver, we use a parallel implementation of the conjugate gradient scheme preconditioned with AMG, provided by DUNE-ISTL. Both of them have their own internal domain decomposition.

We shortly discuss one Estimate-Solve cycle of the MLMC algorithm. Without loss of generality we assume 3-level MLMC. Suppose that we have already computed the required number of samples per level (i.e., we are after Estimate and before Solve). Let us denote by $N_i, i = \{0, 1, 2\}$ the number of required realizations per level for $\widehat{Y}_l$, by p_i the number of processes allocated per $\widehat{Y}_i$, by $p^g_{l_i}$ the respective group size of processes working on a single realization, by n the number of realizations for each group of levels, with t_i the respective time for solving a single problem once, and finally with p^{total} the total number of available processes. Then we can compute the total CPU time for the current Estimate-Solve cycle as

$$T^{\text{total}}_{\text{CPU}} = N_0 t_0 + N_1 t_1 + N_2 t_2.$$

Ideally each process should take $T^p_{\text{CPU}} = T^{\text{total}}_{\text{CPU}}/p^{\text{total}}$. Dividing the CPU time needed for one $\widehat{Y}_i$ by T^p_{CPU}, we get a continuous value for the number of processes on a given level $p^{\text{ideal}}_i = N_i t_i / T^p_{\text{CPU}}$ for $i = \{0, 1, 2\}$. Then we can take $p_i = \lfloor p^{\text{ideal}}_i \rfloor$. To obtain an integer value for the number of processes allocated per level, first we construct a set of all possible splits of the original problem as a combination of subproblems (e.g., parallelize level 2 separately and the combination of levels 0 and 1, or parallelize all levels simultaneously, etc.). Each element of this set is evaluated independently, and all combinations of upper and lower bounds are calculated, such that p^{ideal}_i is divisible by $p^g_{l_i}$, $\sum_{l=0}^{2} p_i < p^{\text{total}}$ and $p_i \leq N_i p^g_{l_i}$. Traversing, computing and summing the computational time needed for each element gives us a time estimation. Then we select the element (grouping of levels) with minimal computational time. To tackle the distribution of the work on a single level, a similar approach can be employed. Due to the large dimension of the search tree a heuristic technique can be employed. Here we consider a simple predefined group size for each deterministic problem, having in mind that when using AMG the work for a single realization at the different levels is proportional to the unknowns at this level.

Numerical Experiments Results for a typical example are shown in Fig. 15. The parameters are $\sigma = 2.75, \lambda = 0.3$. The tests are done on SuperMUC-NG, LRZ Munich on a dual Intel Skylake Xeon Platinum 8174 node. Due to the stochasticity of the problem, we plot the speedup multiplied with the proportion of the tolerance. The renormalization has shown to be a very good approach for defining the coarser levels in MLMC. The proposed parallelization algorithm gives promising scalability results. It is weakly coupled to the number of samples that MLMC estimates.

www.dbooks.org

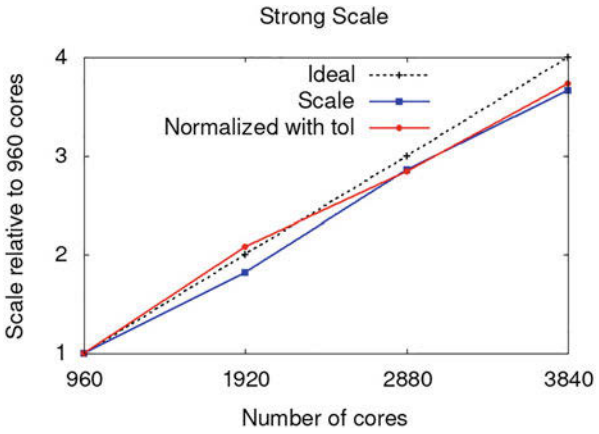


Fig. 15 Scalability of the MLMC approach

Although the search for an optimal solution is an NP-hard problem, the small number of levels enables a full traversing of the search tree. It can be further improved by automatically selecting the number of processes per group that solves a single problem. One also may consider introducing interrupts between the MPI communicators on a level to further improve the performance.

6 Land-Surface Flow Application

To test some of the approaches developed in the EXA-DUNE project, especially the usage of sum-factorized operator evaluation with more complex problems, we developed an application to simulate coupled surface-subsurface flow for larger geographical areas. This is a topic with high relevance for a number of environmental questions from soil protection and groundwater quality up to weather and climate prediction.

One of the central aims of the new approach developed in the project is to be able to relate a physical meaning to the parameter functions used in each grid cell. This is not possible with the traditional structured grid approaches as the necessary resolution would be prohibitive. To avoid the excessive memory requirements of completely unstructured grids we build on previous results for block-structured meshes and use a two-dimensional unstructured grid on the surface which is extruded in a structured way in vertical direction. However, more flexible discretization schemes are needed for such grids, compared to the usual cell-centered Finite-Volume approaches.

6.1 Modelling and Numerical Approach

To describe subsurface flow we use the Richards equation [52]

$$\frac{\partial \theta(\psi)}{\partial t} - \nabla \cdot \left[k(\psi) \left(\nabla \psi + \mathbf{e}_g \right) \right] + q_w = 0$$

where θ is the volumetric water content, ψ the soil water potential, k the hydraulic conductivity, $\mathbf{e}_g$ the unit vector pointing in the direction of gravity and q_w a volumetric source or sink term.

In nearly all existing models for coupled surface-subsurface flow, the kinematic-wave approximation is used for surface flow, which only considers surface slope as driving force and does not even provide a correct approximation of the steady-state solution. The physically more realistic shallow-water-equations are used rarely, as they are computationally expensive. We use the diffusive-wave approximation, which still retains the effects of water height on run-off, yields a realistic steady-state solution and is a realistic approximation for flow on vegetation covered ground [2]:

$$\frac{\partial h}{\partial t} - \nabla \cdot [D(h, \nabla h) \nabla (h + z)] = f_c, \tag{4}$$

where h is the height of water over the surface level z, f_c is a source-sink term (which is used for the coupling) and the diffusion coefficient D is given by

$$D(h, \nabla h) = \frac{h^\alpha}{C \cdot \|\nabla (h + z)\|^{1-\gamma}}$$

with $\| \cdot \|$ refering to the Euclidean norm and α, γ and C are empirical constants. In the following we use $\alpha = \frac{5}{3}$ and $\gamma = \frac{1}{2}$ to obtain Manning's formula and a friction coefficient of $C = 1$.

Both equations are discretised with a cell-centered Finite-Volume scheme and alternatively with a SWIPDG scheme in space (see Sect. 3) and an appropriate diagonally implicit Runge–Kutta scheme in time for the subsurface and an explicit Runge–Kutta scheme for the surface flow. Upwinding is used for the calculation of conductivity in subsurface flow [5] and for the water height in the diffusion term in surface flow.

First tests have shown that the formulation of the diffusive-wave approximation from the literature as given by Eq. (4) does not result in a numerically stable solution if the gradient becomes very small, as then a gradient approaching zero is multiplied by a diffusion coefficient going to infinity. A much better behaviour is achieved by rewriting the equation as

$$\frac{\partial h}{\partial t} - \nabla \cdot \left[\frac{h^\alpha}{C} \cdot \frac{\nabla (h + z)}{\|\nabla (h + z)\|^{1-\gamma}} \right] = f_c,$$

www.dbooks.org

where the rescaled gradient $\frac{\nabla(h+z)}{\|\nabla(h+z)\|^{1-\gamma}}$ is always going to zero when $\nabla(h+z)$ is going to zero as long as $\gamma < 1$ and the new diffusion coefficient h^{α}/C is bounded.

Due to the very different time-scales for surface and subsurface flow, an operator-splitting approach is used for the coupled system. A new coupling condition has been implemented, which is a kind of Dirichlet-Neumann coupling, but guarantees a mass-conservative solution. With a given height of water on the surface (from the initial condition or the last time step modified by precipitation and evaporation), subsurface flow is calculated with a kind of Signorini boundary condition, where all surface water is infiltrated in one time step as long as the necessary gradient is not larger than the pressure resulting from the water ponding on the surface (in infiltration) and potential evaporation rates are maintained as long as the pressure at the surface is not below a given minimal pressure (during evaporation). The advantage of the new approach is that it does not require a tracking of the sometimes complicated boundary between wet and dry surface elements, that it yields no unphysical results and that the solution is mass-conservative even if not iterated until convergence.

Parallelisation is obtained by an overlapping or non-overlapping domain-decomposition (depending on the grid). However, only the two-dimensional surface grid is partitioned whereas the vertical direction is kept on one process due to the strong coupling. Thus there is also no need for communication of surface water height for the coupling, as the relevant data is always stored in the same process. The linear equation systems are solved with the BiCGstab-solver from DUNE-ISTL with Block-ILU0 as preconditioner. The much larger mesh size in horizontal direction compared to the vertical direction results in strong coupling of the unknowns in the vertical direction. The Block-ILU0 scheme provides an almost exact solver of the strongly coupled blocks in the vertical direction and is thus a very effective scheme. Furthermore, one generally has a limited number of cells in the vertical direction and extends the mesh in horizontal direction to simulate larger regions. Thus the good properties of the solver are retained when scaling up the size of the system.

6.2 Performance Optimisations

As the time steps in the explicit scheme for surface flow can get very small due to the stability limit, a significant speedup can be achieved by using a semi-implicit scheme, where the non-linear coefficients are calculated with the solution from the previous time step or iteration. However, if the surface is nearly completely dry, this could lead to numerical problems, thus an explicit scheme is still used under nearly dry conditions with an automatic switching between both.

While matrix-free DG solvers with sum-factorization can yield excellent per node performance (Sect. 3.3), it is a rather tedious task to implement them for new partial differential equations. Therefore, a code-generation framework is currently being developed in a project related to EXA-DUNE [33]. The framework is used to

implement an alternative optimized version of the solver for the Richards equation as this is the computationally most expensive part of the computations. Expensive material functions like the van Genuchten model including several power functions are replaced by cubic spline approximations, which allow a fast vectorized incorporation of flexible material functions to simulate strongly heterogeneous systems. Sum-factorisation is used in the operator evaluations for the DG-discretization with a selectable polynomial degree.

A special pillar grid has been developed as a first realisation of a 2.5D grid [33]. It adds a vertical dimension to a two-dimensional grid (which is either structured or unstructured). However, as the current version still produces a full three-dimensional mesh at the moment, future developments are necessary to exploit the full possibilities of the approach.

6.3 Scalability and Performance Tests

Extensive tests covering infiltration as well as exfiltration have been performed (e.g. Fig. 16) to test the implementation and the new coupling condition. Good scalability is achieved in strong as well as in weak scaling experiments on up to 256 nodes and 4096 cores of the bwForCluster in Heidelberg (Fig. 17). Simulations for a large region with topographical data taken from a digital elevation model (Fig. 18) have been conducted as well.

With the generated code-based solver for the Richards equation a substantial fraction of the system's peak performance (up to 60% on a Haswell-CPU) can be

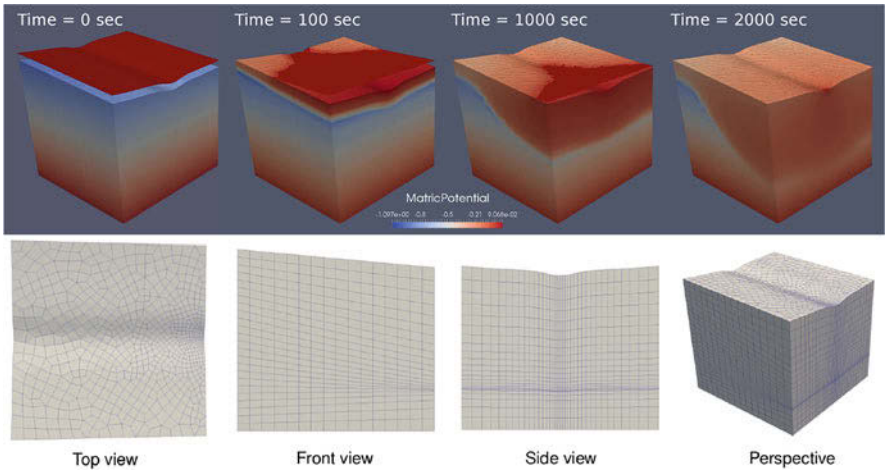


Fig. 16 Surface runoff and infiltration of 5 cm water into a dry coarse sand (top) and the unstructured 2.5D mesh used for the simulations (bottom)

www.dbooks.org

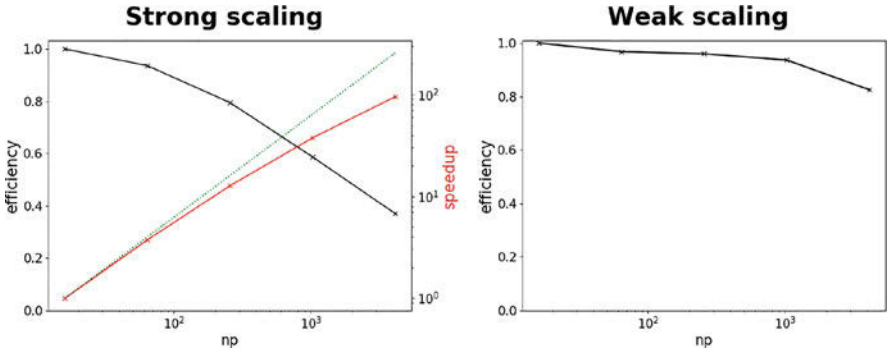


Fig. 17 Results of a strong (left) and weak (right) scalability test with a coupled run-off and infiltration experiment on 1 to 256 nodes (16 to 4096 Intel Xeon E5-2630 v3 2.4 GHz CPU cores) of the bwForCluster at IWR in Heidelberg

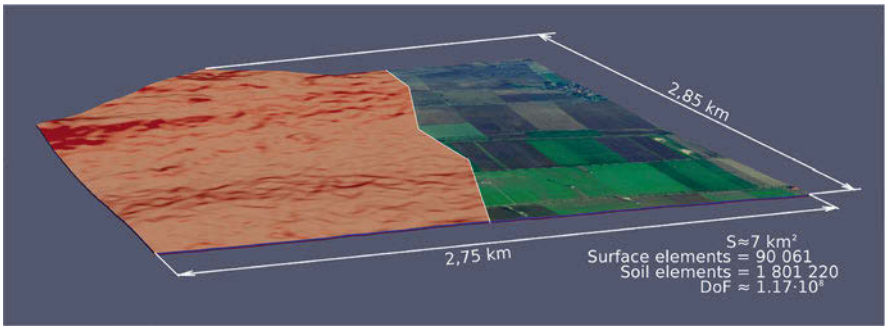


Fig. 18 Pressure distribution with an overlay of the landscape taken from Google Earth calculated in a simulation of water transport in a real landscape south of Brunswick simulated on 30 nodes with 1200 cores of HLRN-IV in Göttingen (2× Intel Skylake Gold 6148 2.4 GHz CPUs)

utilized due to the combination of sum factorization and vectorisation (Fig. 19). For the Richards equation (as for other PDEs before) the number of millions of degrees of freedom per second is independent of the polynomial degree with this approach. We measure a total speedup of 3 compared to the naive implementation in test simulations on a Intel Haswell Core i7-5600U 2.6 GHz CPU with first order DG base functions on a structured $32 \times 32 \times 32$ mesh for subsurface and 32×32 grid for surface flow. Even higher speedups are expected with higher-order base functions and matrix-free iterative solvers. The fully-coupled combination of the Richards solver obtained with the code generation framework and surface-runoff is tested with DG up to fourth order on structured as well as on unstructured grids. Parallel simulations are possible as well.

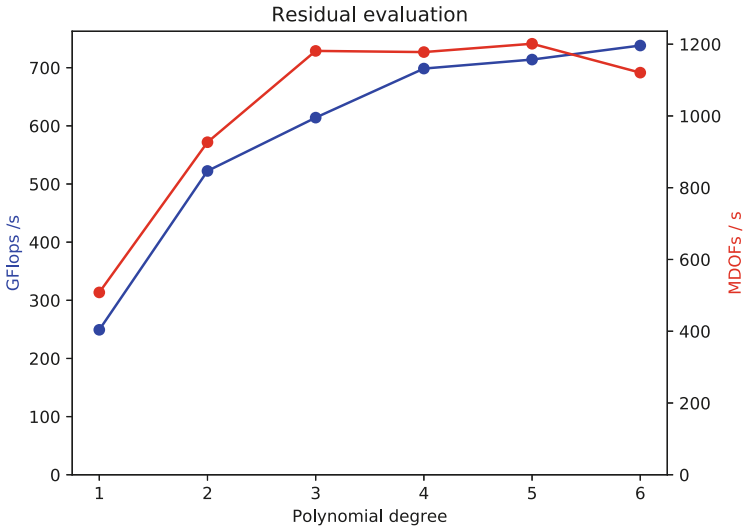


Fig. 19 Performance of the Richards solver implemented with the code generation framework for EXA-DUNE

7 Conclusion

In EXA-DUNE we extended the DUNE software framework in several directions to make it ready for the exascale architectures of the future which will exhibit a significant increase in node level performance through massive parallelism in form of cores and vector instructions. Software abstractions are now available that enable asynchronicity as well as parallel exception handling and several use cases for these abstractions have been demonstrated in this paper: resilience in multigrid solvers and several variants of asynchronous Krylov methods. Significant progress has been achieved in hardware-aware iterative linear solvers: we developed preconditioners for the GPU based on approximate sparse inverses, developed matrix-free operator application and preconditioners for higher-order DG methods and our solvers are now able to vectorize over multiple right hand sides. These building blocks have then been used to implement adaptive localized reduced basis methods, multilevel Monte-Carlo methods and a coupled surface-subsurface flow solver on up to 14k cores. The EXA-DUNE project has spawned a multitude of research projects, running and planned, as well as further collaborations in each of the participating groups. We conclude that the DUNE framework has made a major leap forward due to the EXA-DUNE project and work on the methods investigated here will continue in future research projects.

www.dbooks.org

References

1. Abadi, M.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). http://tensorflow.org/
2. Alonso, R., Santillana, M., Dawson, C.: On the diffusive wave approximation of the shallow water equations. Eur. J. Appl. Math. **19**(5), 575–606 (2008)
3. Amestoy, P.R., Duff, I.S., Koster, J., J.-Y., L.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. **23**(1), 15–41 (2001)
4. Ascher, D., Dubois, P., Hinsen, K., Hugunin, J., Oliphant, T.: Numerical Python. Lawrence Livermore National Laboratory, Livermore (1999)
5. Bastian, P.: A fully-coupled discontinuous Galerkin method for two-phase flow in porous media with discontinuous capillary pressure. Computat. Geosci. **18**(5), 779–796 (2014)
6. Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Kornhuber, R., Ohlberger, M., Sander, O.: A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE. Computing **82**(2–3), 121–138 (2008). https://doi.org/10.1007/s00607-008-0004-9
7. Bastian, P., Blatt, M., Dedner, A., Engwer, C., Klöfkorn, R., Ohlberger, M., Sander, O.: A generic grid interface for parallel and adaptive scientific computing. Part I: abstract framework. Computing **82**(2–3), 103–119 (2008). https://doi.org/10.1007/s00607-008-0003-x
8. Bastian, P., Blatt, M., Scheichl, R.: Algebraic multigrid for discontinuous Galerkin discretizations of heterogeneous elliptic problems. Numer. Linear Algebra Appl. **19**(2), 367–388 (2012). https://doi.org/10.1002/nla.1816
9. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Turek, S.: Advances concerning multiscale methods and uncertainty quantification in EXA-DUNE. In: Software for Exascale Computing—SPPEXA 2013–2015. Lecture Notes in Computational Science and Engineering. Springer Berlin (2016)
10. Bastian, P., Müller, E., Müthing, S., Piatkowski, M.: Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. J. Comput. Phys. **394**, 417–439 (2019). https://doi.org/10.1016/j.jcp.2019.06.001
11. Bland, W., Bosilca, G., Bouteiller, A., Herault, T., Dongarra, J.: A proposal for user-level failure mitigation in the MPI-3 standard. Department of Electrical Engineering and Computer Science, University of Tennessee (2012)
12. Buis, P., Dyksen, W.: Efficient vector and parallel manipulation of tensor products. ACM Trans. Math. Softw. **22**(1), 18–23 (1996). https://doi.org/10.1145/225545.225548
13. Cantwell, C., Nielsen, A.: A minimally intrusive low-memory approach to resilience for existing transient solvers. J. Sci. Comput. **78**(1), 565–581 (2019). https://doi.org/10.1007/s10915-018-0778-7
14. Christie, M., Blunt, M.: Tenth SPE comparative solution project: a comparison of upscaling techniques. In: SPE Reservoir Simulation Symposium (2001). https://doi.org/10.2118/72469-PA
15. Chronopoulos, A., Gear, C.W.: s-step iterative methods for symmetric linear systems. J. Comput. Appl. Math. **25**(2), 153–168 (1989)
16. Dongarra, J., Hittinger, J., Bell, J., Chacón, L., Falgout, R., Heroux, M., Hovland, P., Ng, E., Webster, C., Wild, S., Pao, K.: Applied mathematics research for exascale computing. Tech. rep., U. S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program (2014). http://science.energy.gov/~/media/ascr/pdf/research/am/docs/EMWGreport.pdf
17. Elfverson, D., Ginting, V., Henning, P.: On multiscale methods in Petrov–Galerkin formulation. Numer. Math. **131**(4), 643–682 (2015). https://doi.org/10.1007/s00211-015-0703-z
18. Engwer, C., Dreier, N.A., Altenbernd, M., Göddeke, D.: A high-level C++ approach to manage local errors, asynchrony and faults in an MPI application. In: Proceedings of 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2018) (2018)

19. Engwer, C., Müthing, S.: Concepts for flexible parallel multi-domain simulations. In: Dickopf, T., Gander, M.J., Halpern, L., Krause, R., Pavarino, L.F. (eds.) Domain Decomposition Methods in Science and Engineering XXII, pp. 187–195. Springer, Berlin (2016)
20. Ern, A., Stephansen, A., Vohralík, M.: Guaranteed and robust discontinuous Galerkin a posteriori error estimates for convection-diffusion-reaction problems. J. Comput. Appl. Math. **234**(1), 114–130 (2010). https://doi.org/10.1016/j.cam.2009.12.009
21. Ern, A., Stephansen, A., Zunino, P.: A discontinuous Galerkin method with weighted averages for advection-diffusion equations with locally small and anisotropic diffusivity. IMA J. Numer. Anal. **29**(2), 235–256 (2009). https://doi.org/10.1093/imanum/drm050
22. Fehn, N., Wall, W., Kronbichler, M.: A matrix-free high-order discontinuous Galerkin compressible Navier–Stokes solver: a performance comparison of compressible and incompressible formulations for turbulent incompressible flows. Numer. Methods Fluids **89**(3), 71–102 (2019)
23. Fog, A.: VCL C++ vector class library (2017). http://www.agner.org/optimize/vectorclass.pdf
24. Fritze, R., Leibner, T., Schindler, F.: dune-gdt (2016). https://doi.org/10.5281/zenodo.593009
25. Gagliardi, F., Moreto, M., Olivieri, M., Valero, M.: The international race towards exascale in Europe. CCF Trans. High Perform. Comput. **1**(1), 3–13 (2019). https://doi.org/10.1007/s42514-019-00002-y
26. Geveler, M., Ribbrock, D., Göddeke, D., Zajac, P., Turek, S.: Towards a complete FEM-based simulation toolkit on GPUs: unstructured grid finite element geometric multigrid solvers with strong smoothers based on sparse approximate inverses. Comput. Fluids **80**, 327–332 (2013). https://doi.org/10.1016/j.compfluid.2012.01.025
27. Ghysels, P., Vanroose, W.: Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. Parallel Comput. **40**(7), 224–238 (2014)
28. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. J. Comput. Sci. **17**, 509–521 (2016). https://doi.org/10.1016/j.jocs.2016.06.006
29. Göddeke, D., Altenbernd, M., Ribbrock, D.: Fault-tolerant finite-element multigrid algorithms with hierarchically compressed asynchronous checkpointing. Parallel Comput. **49**, 117–135 (2015). https://doi.org/10.1016/j.parco.2015.07.003
30. Gupta, S., Patel, T., Tiwari, D., Engelmann, C.: Failures in large scale systems: long-term measurement, analysis, and implications. In: Proceedings of the 30th IEEE/ACM International Conference on High Performance Computing, Networking, Storage and Analysis (SC) (2017)
31. Huber, M., Gmeiner, B., Rüde, U., Wohlmuth, B.: Resilience for massively parallel multigrid solvers. SIAM J. Sci. Comput. **38**(5), S217–S239
32. Iliev, O., Mohring, J., Shegunov, N.: Renormalization based MLMC method for scalar elliptic SPDE. In: International Conference on Large-Scale Scientific Computing, pp. 295–303 (2017)
33. Kempf, D.: Code generation for high performance PDE solvers on modern architectures. Ph.D. thesis, Heidelberg University (2019)
34. Kempf, D., Heß, R., Müthing, S., Bastian, P.: Automatic code generation for high-performance discontinuous Galerkin methods on modern architectures (2018). arXiv:1812.08075
35. Keyes, D.: Exaflop/s: the why and the how. Comptes Rendus Mécanique **339**(2–3), 70–77 (2011). https://doi.org/10.1016/j.crme.2010.11.002
36. Klawonn, A., Köhler, S., Lanser, M., Rheinbach, O.: Computational homogenization with million-way parallelism using domain decomposition methods. Comput. Mech. **65**(1), 1–22 (2020). https://doi.org/10.1007/s00466-019-01749-5
37. Kohl, N., Thönnes, D., Drzisga, D., Bartuschat, D., Rüde, U.: The HyTeG finite-element software framework for scalable multigrid solvers. Int. J. Parallel Emergent Distrib. Syst. **34**(5), 477–496 (2019). https://doi.org/10.1080/17445760.2018.1506453
38. Kretz, M., Lindenstruth, V.: Vc: A C++ library for explicit vectorization. Softw. Practice Exp. **42**(11), 1409–1430 (2012). https://doi.org/10.1002/spe.1149
39. Kronbichler, M., Kormann, K.: A generic interface for parallel cell-based finite element operator application. Comput. Fluids **63**, 135–147 (2012). https://doi.org/10.1016/j.compfluid.2012.04.012

www.dbooks.org

40. Lengauer, C., Apel, S., Bolten, M., Chiba, S., Rüde, U., Teich, J., Größlinger, A., Hannig, F., Köstler, H., Claus, L., Grebhahn, A., Groth, S., Kronawitter, S., Kuckuk, S., Rittich, H., Schmitt, C., Schmitt, J.: ExaStencils: Advanced multigrid solver generation. In: Software for Exascale Computing—SPPEXA 2nd phase. Springer, Berlin (2020)
41. Liang, X., Di, S., Tao, D., Li, S., Li, S., Guo, H., Chen, Z., Cappello, F.: Error-controlled lossy compression optimized for high compression ratios of scientific datasets. In: IEEE Big Data, pp. 438–447 (2018). https://doi.org/10.1109/BigData.2018.8622520
42. Lu, Y.: Paving the way for China exascale computing. CCF Trans. High Perform. Comput. **1**(2), 63–72. (2019). https://doi.org/10.1007/s42514-019-00010-y
43. Luporini, F., Ham, D.A., Kelly, P.H.J.: An algorithm for the optimization of finite element integration loops. ACM Trans. Math. Softw. **44**(1), 3:1–3:26 (2017). https://doi.org/10.1145/3054944
44. Message Passing Interface Forum: MPI: a message-passing interface standard version 3.1 (2015). https://www.mpi-forum.org/docs/
45. Milk, R., Rave, S., Schindler, F.: pyMOR—generic algorithms and interfaces for model order reduction. SIAM J. Sci. Comput. **38**(5), S194–S216 (2016). https://doi.org/10.1137/15M1026614
46. Milk, R., Schindler, F., Leibner, T.: Extending DUNE: the dune-xt modules. Arch. Numer. Softw. **5**(1), 193–216 (2017). https://doi.org/10.11588/ans.2017.1.27720
47. Mohring, J., Milk, R., Ngo, A., Klein, O., Iliev, O., Ohlberger, M., Bastian: uncertainty quantification for porous media flow using multilevel Monte Carlo. In: International Conference on Large-Scale Scientific Computing, pp. 145–152 (2015)
48. Müthing, S., Piatkowski, M., Bastian, P.: High-performance implementation of matrix-free high-order discontinuous Galerkin methods (2017). arXiv:1711.10885
49. Ohlberger, M., Schindler, F.: Error control for the localized reduced basis multiscale method with adaptive on-line enrichment. SIAM J. Sci. Comput. **37**(6), A2865–A2895 (2015). https://doi.org/10.1137/151003660
50. Orszag, S.: Spectral methods for problems in complex geometries. J. Comput. Phys. **37**(1), 70–92 (1980). https://doi.org/10.1016/0021-9991(80)90005-4
51. Piatkowski, M., Müthing, S., Bastian, P.: A stable and high-order accurate discontinuous Galerkin based splitting method for the incompressible Navier–Stokes equations. J. Comput. Phys. **356**, 220–239 (2018). https://doi.org/10.1016/j.jcp.2017.11.035
52. Richards, L.A.: Capillary conduction of liquids through porous mediums. Physics **1**, 318–333 (1931)
53. Ruelmann, H., Geveler, M., Turek, S.: On the prospects of using machine learning for the numerical simulation of PDEs: training neural networks to assemble approximate inverses. Eccomas Newsletter 27–32 (2018)
54. Teranishi, K., Heroux, M.: Toward local failure local recovery resilience model using MPI-ULFM. In: Proceedings of the 21st European MPI Users' Group Meeting, EuroMPI/ASIA '14, pp. 51:51–51:56 (2014). https://doi.org/10.1145/2642769.2642774
55. Turek, S., Göddeke, D., Becker, C., Buijssen, S., Wobker, S.: FEAST—realisation of hardware-oriented numerics for HPC simulations with finite elements. Concurrency Comput. Pract. Exp. **22**(6), 2247–2265 (2010)
56. Vohralík, M.: A posteriori error estimates for lowest-order mixed finite element discretizations of convection-diffusion-reaction equations. SIAM J. Numer. Anal. **45**(4), 1570–1599 (2007). https://doi.org/10.1137/060653184
57. van der Walt, S., Colbert, S.C., Varoquaux, G.: The NumPy array: a structure for efficient numerical computation. Comput. Sci. Eng. **13**(2), 22–30 (2011). https://doi.org/10.1109/MCSE.2011.37
58. Xu, J.: Iterative methods by space decomposition and subspace correction. SIAM Rev. **34**(4), 581–613 (1992). https://doi.org/10.1137/1034116

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

ExaFSA: Parallel Fluid-Structure-Acoustic Simulation



Florian Lindner, Amin Totounferoush, Miriam Mehl, Benjamin Uekermann, Neda Ebrahimi Pour, Verena Krupp, Sabine Roller, Thorsten Reimann, Dörte C. Sternel, Ryusuke Egawa, Hiroyuki Takizawa, and Frédéric Simonis

Abstract In this paper, we present results of the second phase of the project ExaFSA within the priority program SPP1648—Software for Exascale Computing. Our task was to establish a simulation environment consisting of specialized highly efficient and scalable solvers for the involved physical aspects with a particular focus on the computationally challenging simulation of turbulent flow and propagation of the induced acoustic perturbations. These solvers are then coupled in a modular, robust, numerically efficient and fully parallel way, via the open source coupling library preCICE. Whereas we made a first proof of concept for a three-field simulation (elastic structure, surrounding turbulent acoustic flow in the near-field, and pure acoustic wave propagation in the far-field) in the first phase, we removed several scalability limits in the second phase. In particular, we present new contributions to (a) the initialization of communication between processes of

F. Lindner · A. Totounferoush · M. Mehl (✉)
University of Stuttgart, Stuttgart, Germany
e-mail: florian.lindner@ipvs.uni-stuttgart.de; amin.totounferoush@ipvs.uni-stuttgart.de; miriam.mehl@ipvs.uni-stuttgart.de

B. Uekermann
Eindhoven University of Technology, Eindhoven, Netherlands
e-mail: b.w.uekermann@tue.nl

N. Ebrahami Pour · V. Krupp · S. Roller
University of Siegen, Siegen, Germany
e-mail: neda.epour@uni-siegen.de; sabine.roller@uni-siegen.de

T. Reimann · D. C. Sternel
Technische Universität Darmstadt, Darmstadt, Germany
e-mail: thorsten.reimann@sc.tu-darmstadt.de; doerte.sternel@hpc-hessen.de

R. Egawa · H. Takizawa
Tohoku University, Sendai, Japan
e-mail: takizawa@tohoku.ac.jp

F. Simonis
Technical University of Munich, München, Germany
e-mail: simonis@in.tum.de

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_10

the involved independent solvers, (b) optimization of the parallelization of data mapping, (c) solver-specific white-box data mapping providing higher efficiency but less flexibility, (d) portability and scalability of the flow and acoustic solvers FASTEST and Ateles on vector architectures by means of code transformation, (e) physically correct information transfer between near-field acoustic flow and far-field acoustic propagation.

1 Introduction

The simulation of fluid-structure-acoustic interactions is a typical example for multi-physics simulations. Two fundamentally different physical sound sources can be distinguished: structural noise and flow-induced noise. As we are interested in accurate results for the resulting sound emissions induced from the turbulent flow, it is decisive to include not only the turbulent flow, but also the structure deformation and the interaction between both. High accuracy requires the use of highly resolved grids. As a consequence, the use of massively parallel supercomputers is inevitable. When we are interested in the sound effects far away from a flow induced fluttering structure, the simulation becomes too expensive, even for supercomputing architectures. Hence, we introduce an assumption, we call it the "far-field". Far from the structure and, thus, the noise generation, we assume a homogeneous background flow and restrict the simulation in this part of the domain to the propagation of acoustic waves. This results in an overall setup with two coupling surfaces—between the elastic structure and the surrounding flow, and between the near-field and the far-field in the flow domain (see Fig. 1 for an illustrative example). Such a complex simulation environment implies several new challenges compared to

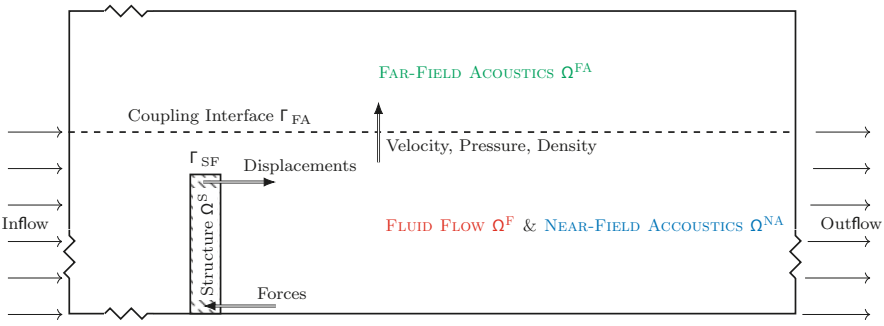


Fig. 1 Multiphysics fluid-structure-acoustic scenario as used in our simulations in Sect. 6. The domain is decomposed into a near-field 'incompressible flow region' $\Omega^{F} = \Omega^{NA}$, a far-field 'acoustic only region' Ω^{FA}, and an 'elastic structure region' Ω^{S}. Note that the geometry is not scaled correctly for better illustration

www.dbooks.org

"single-physics" simulations: (a) multi-scale properties in space and time (small-scale processes around the structure, multi-scale turbulent flow in the near-field, and large-scale processes in the acoustic far-field), (b) different optimal discretization and solver choices for the three fields, (c) highly ill-conditioned problem, if formulated and discretized as a single large system of equations, (d) challenging load-balancing due to different computational load per grid unit depending on the local physics.

Application examples for fluid-structure-acoustic simulations can be found in several technical systems: wind power plants, fans in air conditioning systems of buildings, cars or airplanes, car mirrors and other design details of a car frame, turbines, airfoil design, etc.

Fluid-structure interaction simulations as a sub-problem of our target system have been in the focus of research in computational engineering for many years, mainly aiming at capturing stresses in the structure more realistically than with a pure flow simulation. A main point of discussion in this field is the question whether monolithic approaches—treating the coupled problem as a single large system of equations—or partitioned methods—glueing together separate simulation modules for structures and fluid flow by means of suitable coupling numerics and tools—are more appropriate and efficient. Monolithic approaches require a new implementation of the simulation code as well as the development of specialized iterative solvers for the ill-conditioned overall system of equations, but can achieve very high efficiency and accuracy [3, 12, 19, 23, 38]. Partitioned approaches, on the other hand, offer large flexibility in choosing optimal solvers for each field, adding additional fields, or exchanging solvers. The difficulty here lies in both a stable, accurate, and efficient coupling between independent solvers applying different numerical methods and in establishing efficient communication and load balancing between the used parallel codes. For numerical coupling, numerous efficient data mapping methods [5, 26, 27, 32] have been published along with efficient iterative solvers [2, 7, 13, 20, 29, 35, 39, 41]. In [6], various monolithic and partitioned approaches have been proposed and evaluated in terms of a common benchmark problem. Three-field fluid-structure-acoustic interaction in the literature has so far been restricted to near-field simulations due to the intense computational load [28, 33].

To realize a three-field fluid-structure-acoustic interaction including the far-field, we use a partitioned approach and couple existing established "single-physics" solvers in a black-box fashion. We couple the finite volume solver FASTEST [18], the discontinuous Galerkin solver Ateles [42], and the finite element solver CalculiX [14] by means of the coupling library preCICE [8]. We compare this approach to a less flexible white-box coupling implemented in APESmate [15] as part of the APES framework and make use of the common data-structure within APES [31]. The assumption which is confirmed in this paper is, that the white-box approach is more efficient, but puts some strict requirements on the codes to be coupled, while the black-box approach is a bit less efficient, but much more flexible with respect to

the codes that can be used. Our contributions to the field of fluid-structure-acoustic interaction, which we summarize in this paper, include:

1. For the near-field flow, we introduce a volume coupling between background flow and acoustic perturbations in FASTEST accounting for the multi-scale properties in space and time by means of different spatial and time resolution.
2. For both near-field flow and far-field acoustics, we achieved portability and performance optimization of Ateles and FASTEST for vector machines by means of code transformation.
3. In terms of inter-field coupling, we
   (a) increased the efficiency of inter-code communication by means of a new hierarchical implementation of communication initialization and a modified communicator concept,
   (b) we improved the robustness and efficiency of radial basis function mapping,
   (c) we identified correct interface conditions between near-field and far-field, optimized the position of the interface, and ensured correct boundary conditions by overlapping near-field and far-field,
   (d) we developed and implemented implicit quasi-Newton coupling numerics that allow for a simultaneous execution of all involved solvers.
4. For a substantially improved inter-code load balancing, we use a regression-based performance model for all involved solvers and perform an optimization of assigned cores.
5. We present a comparison of our black-box and to the white-box approach for multi-physics coupling.

These contributions have been achieved as a result of the project ExaFSA—a cooperation between the Technische Universität Darmstadt, the University of Siegen, the University of Stuttgart, and the Tohoku University (Japan) in the Priority Program SPP 1648—Software for Exascale Computing of the German Research Foundation (DFG) in close collaboration with the Technical University of Munich. In the first funding phase (2013–2016), we showed that efficient yet robust coupled simulations are feasible and can be enhanced with an in-situ visualization component as an additional software part, but we still reached limits in terms of scalability and load balancing [4, 9]. This paper focuses on results of the second funding phase (2016–2019) and demonstrates significant improvements in scalability and accuracy as well as robustness based on the above-mentioned contributions.

In the following, we introduce the underlying model equations of our target scenarios in Sect. 2 and present our solvers and their optimization in Sect. 3 as well as the black-box coupling approach and new contributions in terms of coupling in Sect. 4. In Sect. 5, we compare black-box coupling to an alternative, efficient, but solver-specific and, thus, less flexible white-box coupling for uni-directional flow-acoustic coupling. Finally, results for a turbulent flow over a fence scenario are presented in Sect. 6.

www.dbooks.org

2 Model

In this section, we shortly introduce the underlying flow, acoustic and structure models of our target application. We use the Einstein summation convention throughout this section.

2.1 Governing Equations

The multi-physics scenario we investigate describes an elastic structure embedded in a turbulent flow field. The latter is artificially decomposed into a near-field and a far-field. See Fig. 1 for an example.

Near-Field Flow In the near-field region $\Omega^{\mathrm{F}} = \Omega^{\mathrm{NA}}$, the compressible fluid flow is modeled by means of the density $\overline{\rho}$, the velocity $\overline{u_i}$ and the pressure $\overline{p}$. As we focus on a low Mach number regime, we can split these variables into an incompressible part ρ, u_i, p, and acoustic perturbations ρ', u_i', p':

$$\overline{\rho} = \rho + \rho', \quad \overline{u_i} = u_i + u_i', \quad \overline{p} = p + p' . \tag{1}$$

The incompressible flow is described by the Navier-Stokes equations[1]

$$\frac{\partial u_i}{\partial x_i} = 0, \tag{2a}$$

$$\frac{\partial}{\partial t}\left(\rho u_i\right) + \frac{\partial}{\partial x_j}\left(\rho u_i u_j - \tau_{ij}\right) = \rho f_i \,, \tag{2b}$$

where ρ is the density of the fluid, and f_i summarizes external force density terms. The incompressible stress tensor τ_{ij} for a Newtonian fluid is described by

$$\tau_{ij} = -p\delta_{ij} + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) , \tag{3}$$

with μ representing the dynamic viscosity and δ_{ij} the Kronecker-Delta.

[1]To capture the moving structure within the near-field, we actually formulate all near-field equations in an arbitrary Lagrian-Eulerian perspective. For the relative mesh velocity, we use a block-wise elliptic mesh movement as described in [30]. As we do not show fluid-structure interaction in this contribution, however, we formulate all near-field equations in a pure Eulerian perspective for the sake of simplicity.

Acoustic Wave Propagation The propagation of acoustic perturbations in both the near-field and the far-field is modeled by the linearized Euler equations, where in the far-field a constant background state is assumed (which implies $\frac{\partial p}{\partial t} = 0$).

$$\frac{\partial \rho'}{\partial t} + \rho \frac{\partial u_i'}{\partial x_i} + u_i \frac{\partial \rho'}{\partial x_i} = 0 \tag{4a}$$

$$\rho \frac{\partial u_i'}{\partial t} + \rho u_j \frac{\partial u_i'}{\partial x_j} + \frac{\partial p'}{\partial x_i} = 0 \tag{4b}$$

$$\frac{\partial p'}{\partial t} + \rho c^2 \frac{\partial u_i'}{\partial x_i} + u_i \frac{\partial p'}{\partial x_i} = -\frac{\partial p}{\partial t} \tag{4c}$$

Here c is the speed of sound. In the near-field, the background flow quantities u_i and p are calculated from (2), whereas they are assumed to be constant in the acoustic far-field. The respective constant value is read from the coupling interface with the near-field, which implies that the interface has to be chosen such that the background flow values are (almost) constant at the coupling interface. In both cases, the coupling between background-flow and acoustic perturbations is uni-directional from the background flow to the acoustic equations (4) by means of p and u_i.

Elastic Structure The structural subdomain Ω^{S} is governed by the equations of motion, here in Lagrangian description:

$$\rho^{\mathrm{S}} \frac{\partial^2 \vartheta_i}{\partial t^2} = \frac{\partial S_{jk} F_{ik}}{\partial X_j^{\mathrm{S}}} + \rho^{\mathrm{S}} f_i^{\mathrm{S}}. \tag{5}$$

With $x_i^{\mathrm{S}} = X_i^{\mathrm{S}} + \vartheta_i$ being the position of a particle in the current configuration, X_i^{S} is the position of a particle in the reference configuration, and ϑ_i the displacement. F_{ij} is the deformation gradient. S_{ij} is the second Piola-Kirchhoff tensor, and ρ^{S} describes the structural density. The Cauchy stress tensor τ_{ij}^{S} relates to S_{ij} via

$$\tau_{ij}^{\mathrm{S}} = \frac{1}{\det\left(F_{ij}\right)} F_{ik} S_{kl} F_{jl}\,. \tag{6}$$

We assume linear elasticity to describe the stress-strain relation.

The coupling between fluid and structure is bi-directional by means of dynamic and kinematic conditions, i.e., equality of interface displacements/velocities and stresses, i.e.,

$$u_i^{\Gamma^{\mathrm{F}}} = \frac{\partial \vartheta_i^{\Gamma^{\mathrm{S}}}}{\partial t}, \qquad \tau_{ij}^{\Gamma^{\mathrm{F}}} = \tau_{ij}^{\Gamma^{\mathrm{S}}} \tag{7}$$

at $\Gamma^{\mathrm{I}} = \Gamma^{\mathrm{S}} \cap \Gamma^{\mathrm{F}}$ with $\Gamma^{\mathrm{F}} = \partial\Omega^{\mathrm{F}}$ and $\Gamma^{\mathrm{S}} = \partial\Omega^{\mathrm{S}}$.

www.dbooks.org

3 Solvers and Their Optimization

Following a partitioned approach, the respective subdomains of the multi-physics model as described in Sect. 2 (elastic structure domain, near-field, and far-field) are treated by different solvers. We employ the flow solver FASTEST presented in Sect. 3.1 to solve for the incompressible flow equations, Eq. (2), and near-field acoustics equations, Eq. (4), the Ateles solver described in Sect. 3.2 for the far-field acoustics equations, Eq. (4), and finally the structural solver CalculiX introduced in Sect. 3.3 for the deformation of the obstacle, Eq. (5). For performance optimization of FASTEST and Ateles, we make use of the Xevolver framework, which has been developed to separate system-specific performance concerns from application codes. We report on the optimization of both solvers further below.

3.1 FASTEST

FASTEST is used to solve both the incompressible Navier-Stokes equations (2) and the linearized Euler equations (4) in the near-field.

Capabilities and Numerical Methods The flow solver FASTEST [24] solves the three-dimensional incompressible Navier-Stokes equations. The equations are discretized utilizing a second-order finite-volume approach with implicit time-stepping, which is also second order accurate. Field data are evaluated on a non-staggered, body-fitted, and block-structured grid. The equations are solved according to the SIMPLE scheme [11], and the resulting linear equation system is solved by ILU factorization [36]. Geometrical multi-grid is employed for convergence acceleration. The code generally follows a hybrid parallelization strategy employing MPI and OpenMP. FASTEST can account for different flow phenomena, and has the capability to model turbulent flow with different approaches. In our test case example, we employ a detached-eddy simulation (DES) based on the $\zeta - \mathrm{f}$ turbulence model [30].

In addition, FASTEST contains a module to solve the linearized Euler equations to describe low Mach number aeroacoustic scenarios, which are solved by a second order Lax-Wendroff scheme with various limiters.

Since all equation sets are discretized on the same numerical grid, advantage can be taken from the multi-grid capabilities to account for the scale discrepancies of the fluid flow and the acoustics. Since the spatial scales of the acoustics are considerably larger than those of the flow, a coarser grid level can be used for them. In return, the finer temporal scales can be considered by sub-cycling a CFD time step with various CAA time steps. This way a very efficient implementation of the viscous/acoustic splitting approach can be realized.

Performance Optimization Concerning performance optimization, one interesting point of FASTEST is that some of its kernels were once optimized for old vector machines, and thus important kernels have their vector versions in addition to the default ones. The main difference between the two versions is that nested loops in the default version are collapsed into one loop in the vector version. Since the loops skip accessing halo regions, the compiler is not able to automatically collapse the loops, resulting in short vector lengths even if the compiler can vectorize them. To efficiently run the solver on a vector system, performance engineers usually need to manually change the loop structures. In this project, Xevolver is used to express the differences between the vector and default versions as code transformation rules. In other words, vectorization-aware loop optimizations are expressed as code transformations. As a result, the default version can be transformed to its vector version, and the vector version does not need to be maintained any longer to achieve high performance on vector systems. That is, the FASTEST code can be simplified without reducing the vector performance by using the Xevolver approach. Ten rules are defined to transform the default kernels in FASTEST to their vector kernels. Those code transformations plus some system-independent minor code modifications for removing vectorization obstacles can reduce the execution time on the NEC SX-ACE vector system by about 85%, when executing a simple test case that models a three-dimensional Poiseuille flow through a channel based on the Navier-Stokes equations, in which the mesh contains two blocks with 426,000 cells each. The code execution on the SX-ACE vector processor works about 2.7 times faster than on the Xeon E5-2695v2 processor, since the kernel is memory-intensive and the memory bandwidth of SX-ACE is 4× higher than that of Xeon. Therefore, it is clearly demonstrated that the Xevolver approach is effective to achieve both high performance portability and high code maintainability for FASTEST.

3.2 Ateles

In our project, Ateles is used for the simulation of the acoustic far-field. Since acoustics scales need to be transported over a large distance, Ateles' high-order DG scheme can show its particular advantages of low dissipation and dispersion error in this test case.

Capabilities and Numerical Methods The solver Ateles is integrated in the simulation framework APES [31]. Ateles is based on the Discontinuous Galerkin (DG) discretization method, which can be seen as a hybrid method, combining the finite-volume and finite-element methods. DG is well suited for parallelization and the simulation of aero-acoustic problems, due to its inherent dissipation and dispersion properties. This method has several outstanding advantages, that are among others the high-order accuracy, the faster convergence of the solution with

www.dbooks.org

increasing scheme order and fewer elements compared to a low order scheme with a higher number of elements, the local h-p refinement as well as orthogonal hierarchical bases. The DG solver Ateles includes different equation systems, among others the compressible Navier-Stokes equations, the compressible inviscid Euler equations and the linearized Euler equations (used in this work for the acoustics far-field). For the time discretization, Ateles makes use of the explicit Runge-Kutta time stepping scheme, which can be either second or fourth order.

Performance Optimization Analyzing the performance of Ateles, originally developed assuming x86 systems, we found out that four kinds of code optimization techniques are needed for a total of 18 locations of the code in order to migrate the code to the SX-ACE system. Those techniques are mostly for collapsing the kernel loop and also for directing the NEC compiler to vectorize the loop. In this project, all the techniques are expressed as one common code transformation rule. The rule can take the option to change its transformation behaviors appropriately for each code location. This means that, to achieve performance portability between SX-ACE and x86 systems, only one rule needs to be maintained in addition to the Ateles application code. We executed a small testcase solving Maxwell equations with an 8th order DG scheme on 64 grid cells. The code transformation leads to 7.5× higher performance. The significant performance improvement is attributed to loop collapse and insertion of appropriate compiler directives, which increases the vectorization length by a factor of 2 and the vectorization ratio from 71.35% to 96.72%. Finally, in terms of the execution time, the SX-ACE performance is 19% the performance of Xeon E5-2695v2. The code optimizations for SX-ACE reduce the performances of Xeon and Power8 by 14% and 6%, respectively. In this way, code optimizations for a specific system are often harmful to other systems. However, by using Xevolver, such a system-specific code optimization is expressed separately from the application code. Therefore, the Xevolver approach is obviously useful for achieving high performance portability across various systems without complicating the application code.

3.3 CalculiX

As structure solver, we use the well-established finite element solver CalculiX[14], developed by Guido Dhont und Klaus Wittig.[2] While CalculiX also supports static and thermal analysis, we only use it for dynamic non-linear structural mechanics. As our main research focus is not the structural computation per se, but the coupling within a fluid-structure-acoustic framework, we merely regard CalculiX as a black box. The preCICE adapter of CalculiX has been developed in [40].

[2] www.calculix.de.

4 A Black-Box Partitioned Coupling Approach Using preCICE

Our first and general coupling approach for the three-field simulation comprising (a) the elastic structure, (b) the near-field flow with acoustic equations, and (c) the far-field acoustic propagation follows a black-box idea, i.e., we only use input and output data of dedicated solvers at the interfaces between the respective domains for numerical coupling. Such a black-box coupling requires three main functional coupling components: intercode-communication, data-mapping between non-matching grids of independent solvers, and iterative coupling in cases with strong bi-directional coupling. preCICE is an open source library[3] that provides software modules for all three components. In the first phase of the ExaFSA project, we ported preCICE from a server-based to a fully peer-to-peer communication architecture [9, 39], increasing the scalability of the software from moderately to massively parallel. To this end, all coupling numerics needed to be parallelized on distributed data. During the second phase of the ExaFSA project, we focused on several costly initialization steps and further necessary algorithmic optimizations. In the following, we shortly sketch all components of preCICE with a particular focus on innovations introduced in the second phase of the ExaFSA project and on the actual realization of the fluid-acoustic coupling between near-field and far-field and the fluid-structure coupling.

4.1 (Iterative) Coupling

To simulate fluid-structure-acoustic interactions such as in the scenario shown in Fig. 1, two coupling interfaces have to be considered with different numerical and physical properties: (a) the coupling between fluid flow and the elastic structure requires an implicit bi-directional coupling, i.e., we exchange data in both directions and iterate in each time step until convergence; (b) the coupling between fluid flow and the acoustic far-field is uni-directional (neglecting reflections back into the near-field domain), i.e., results of the near-field fluid flow simulation are propagated to the far-field solver as boundary values once per time step. In order to fulfil the coupling conditions at the fluid-structure interface as given in Sect. 2, we iteratively solve the fixed-point equation

$$\begin{pmatrix} S(f) \\ F(u) \end{pmatrix} = \begin{pmatrix} u \\ f \end{pmatrix}, \tag{8}$$

[3] www.precice.org.

www.dbooks.org

where f represents the stresses, u the velocities at the interface Γ_{FS}, S the effects of the structure solver on the interface (with stresses as an input and velocities as an output), F the effects of the fluid solver on the interface (with interface velocities as an input and stresses as an output). preCICE provides a choice of iterative methods accelerating the plain fixed-point iteration on Eq. (8). The most efficient and robust schemes are our quasi-Newton methods that are provided in a linear complexity (in terms of interface degrees of freedom) and fully parallel optimized versions [35]. As most of our achievements concerning iterative methods fall within the first phase of the ExaFSA project, we omit a more detailed description and refer to previous reports instead [9].

For the uni-directional coupling between the fluid flow in the near-field and the acoustic far-field, we transfer perturbation in density, pressure, and velocity from the flow domain to the far-field as boundary conditions at the interface. We do this once per acoustic time step, which is chosen to be the same for near-field and far-field acoustics, but which is much smaller than the fluid time step size (and the fluid-structure coupling), as described in Sect. 3.1.

Both domains are time-dependent and subject to mutual influence.

In an aeroacoustic setting, the near-field subdomain Ω^{NA} and far-field subdomain Ω^{FA}, with boundaries $\Gamma^{\mathrm{NA}} = \partial\Omega^{\mathrm{NA}}$ and $\Gamma^{\mathrm{FA}} = \partial\Omega^{\mathrm{FA}}$ are fixed, which means, all background information in the far-field are fixed to a certain value. Therefore there is only influence of Ω^{NA} onto Ω^{FA}, as backward propagation can be neglected. Then the continuity of shared state variables on the interface boundary $\Gamma^{\mathrm{IA}} = \Gamma^{\mathrm{NA}} \cap \Gamma^{\mathrm{FA}}$ is

$$\rho_i'^{\Gamma^{\mathrm{FA}}} = \rho_i'^{\Gamma^{\mathrm{NA}}}, \; u_i'^{\Gamma^{\mathrm{FA}}} = u_i'^{\Gamma^{\mathrm{NA}}}, \; p_i'^{\Gamma^{\mathrm{FA}}} = p_i'^{\Gamma^{\mathrm{NA}}} \quad . \tag{9}$$

4.2 Data Mapping

Our three solvers use different meshes adapted to their specific problem domain. To map data between the meshes, preCICE offers three different interpolation algorithms: (a) Nearest-neighbor interpolation is based on finding the geometrically nearest neighbor, i.e. the vertex with the shortest distance from the target or source vertex. It excels in its ease of implementation, perfect parallelizability, and low memory consumption. (b) Nearest-projection mapping can be regarded as an extension to the nearest-neighbor interpolation, working on nearest mesh elements (such as edges, triangles or quads) instead of merely vertices and interpolating values to the projection points. The method requires a suitable triangulation to be provided by the solver. (c) Interpolation by radial-basis functions is provided. This method works purely on vertex data and is a flexible choice for arbitrary mesh combinations with overlaps and gaps alike.

In the second phase of the ExaFSA project, we improved the performance of the data mapping schemes in various ways. All three interpolation algorithms contain a lookup-phase which searches for vertices or mesh elements near a given set of

positions. As there is no guarantee regarding ordering of vertices, this resulted in $O(n \cdot m)$ lookup operations, $n, m \in \mathbb{N}$ being the size of the respective meshes. In the second phase, we introduced a tree-based data structure to facilitate efficient spatial queries. The implementation utilizes the library `Boost Geometry`[4] and uses an rtree in conjunction with the `r-star` insertion algorithm. The integration of the tree is designed to fit seamlessly into preCICE and avoids expensive copy operations for vertices and mesh elements of higher dimensionality. Consequently, the complexity of the lookup-phase was reduced to $O\left(\log_a n\right) \cdot m$ with a being a parameter of the tree, set to $\approx$5. The tree index is used by nearest-neighbor, nearest-projection, and RBF interpolation as well as other parts in preCICE and provides a tremendous speedup in the initialization phase of the simulation.

In the course of integrating the index, the RBF interpolation profited from a second performance improvement. In contrast to the nearest-neighbor and nearest-projection schemes it creates an explicit interpolation matrix. Setting values one by one results in a large number of small memory allocations with a relatively large per-call overhead. To remedy this, a preallocation pattern is computed with the help of the tree index. This results in a single memory allocation, speeding up the process of filling the matrix. A comparison of the accuracy and runtime of the latter two interpolation methods is provided in Sect. 5.

4.3 Communication

Smart and efficient communication is paramount in a partitioned multi-physics scenario. As preCICE is targeted at HPC systems, a central communication instance would constitute a bottleneck and has to be avoided. At the end of phase one, we implemented a distributed application architecture. The main objective in its design is not a classical speed-up (as it is for parallelism) but not to deteriorate the scalability of the solvers and rendering a central instance unnecessary. Still, a so-called *master* process exists, which has a special purpose mainly during the initialization phase.

At initialization time, each solver gives its local portion of the interface mesh to preCICE. By a process called re-partitioning, the mesh is transferred to the coupling partner and partitioned there, i.e., the coupling partner's processes select interface data portions that are relevant for their own calculations. The partitioning pattern is determined by the requirements of the selected mapping scheme. The outcome of this process is a sparse communication graph, where only links between participants exist that share a common portion of the interface. While this process was basically in place at the end of phase one, it was refined in several ways.

MPI connections are managed by means of a communicator which represents an n-to-m connection including an arbitrary number of participants. The first imple-

[4] www.boost.org.

www.dbooks.org

mentation used only one communication partner per communicator, essentially creating only 1-to-1 connections. To establish the connections, every connected pair of ranks had to exchange a connection token generated by the accepting side. This exchange is performed using the network file system, as the only a-priori existing communication space common to both participants. However, network file systems tend to perform badly with many files written to a single directory. To reduce the load on the file system, a hash-based scheme was introduced as part of the optimizations in phase two. With that, writing of the files is distributed among several directories, as presented in [26]. This scheme features a uniform distribution of files over different directories and, thus, minimizes the files per directory.

However, this obviously resulted in a large number of communicators to be created. As a consequence, large runs hit system limits regarding the number of communicators. Therefore, a new MPI communication scheme was created as an alternative. It uses only one communicator for an all-to-all communication, resulting in significant performance improvements for the generation of the connections. This approach also solves the problem of the high number of connection tokens to be published, though only for MPI. As MPI is not always available or the implementation is lacking, the hash-based scheme of publishing connection tokens is still required for TCP based connections.

4.4 Load Balancing

In a partitioned coupled simulation solvers need to exchange boundary data at the beginning of each iteration, which implies a synchronization point. If computational cores are not distributed in an optimal way among solvers, one solver will have to wait for the other one to finish its time step. Thus, the load imbalance reduces the computational performance. In addition, in a one way coupling scenario, if the data receiving solver is much slower than the other one, the sending partner has to wait until the other one is ready to receive (in synchronized communication) or store the data in a buffer (in asynchronous communication). In the first phase, the distribution of cores over solvers was adjusted manually and only synchronized communication was implemented, resulting in idle times.

Regression Based Load Balancing We use the load balancing approach proposed in [37] to find the optimal core distribution among solvers: we first model the solver performance against the number of cores for each domain and then optimize the core distribution to minimize the waiting time. Since mathematical modeling of the solvers' performance can be very complicated, we use an empirical approach as proposed in [37], first introduced in [10], to find an appropriate model.

Assuming we have a given set of m data points, consisting of pairs (p, f_p) mapping the number of ranks p to the run-time f_p, we want to find a function $f(p)$ which predicts the run-time against p. Therefore, we use the Performance Model

Normal Form (PMNF) [10] as a basis for our prediction model:

$$f^i(p) = \sum_{k=1}^{n} c_k p^{i_k} \log_2^{j_k}(p), \tag{10}$$

where the superscript i denotes the respective solver, n is a-priori chosen number of terms, $i_k, j_k \in \mathbb{N}_0$ and c_k is the coefficient for the kth regression term. The next step is to optimize the core distribution such that we achieve minimal overall run time which can be expressed by the following optimization problem:

$$\begin{aligned} &\underset{p_1,\ldots,p_l}{\text{minimize}} \quad F(p_1,\ldots,p_l) \quad \text{with } F(p_1,\ldots p_l) = \max_i(f^i(p_i)) \\ &\text{subject to} \quad \sum_{i=1}^{l} p_i \leq P. \end{aligned}$$

This optimization problem is a nonlinear, possibly non-convex integer program. It can be solved by the use of branch and bound techniques. But, if we assume that the f^i are all monotonically decreasing, i.e., assigning more cores to a solver never increases the run-time, we can simplify the constraints to $P = \sum_{i=0}^{l} p_i$ and solve the problem by brute-forcing all possible choices for p_i. That is, we iterate over all possible combinations of core numbers and choose the pair that minimizes the total run-time. For more details, please refer to [37].

Asynchronous Communication and Buffering For our fluid-structure-acoustic scenario shown in Fig. 1, we perform an implicitly coupled simulation of the elastic structure interacting with the incompressible flow over a given discrete time step (marked simply as 'Fluid' in Fig. 2). This is followed by many small time steps for the acoustic wave propagation in the near-field, which are coupled in a loose, unidirectional way to the far-field acoustic solver (executing the same small time steps). To avoid waiting times of the far-field solver while we compute the fluid-structure interactions in the near-field, we would like to 'stretch' the far-field calculations such that they consume the same time as the sum of fluid-structure time steps and acoustic steps in the near-field (see Fig. 2). To achieve this, we introduced a fully asynchronous buffer layer, by which the sending participant was decoupled from the receiving participant, as shown in Fig. 2. Special challenges to tackle were the preservation of the correct ordering of messages, especially for TCP communication which does not implement such guarantees in the protocol.

4.5 Isolated Performance of preCICE

In this section, we show numerical results for preCICE only. This isolated approach is used to show the efficiency of the communication initialization. In addition,

www.dbooks.org

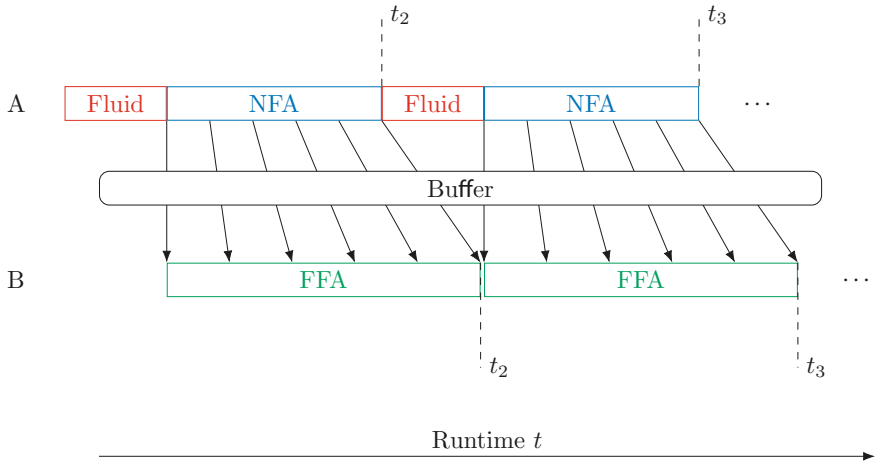


Fig. 2 Coupling scenario between participant A (performing a time step for the incompressible fluid (or fluid-structure interaction) followed by many time steps of the near-field acoustic simulation (NFA)) and participant B (performing the same small acoustic steps for the far-field (FFA) after receiving acoustic data from the near-field solver). Without buffering, inevitable idle times for participant B are created. NFA is linked to FFA through send operations. Therefore, the runtimes of NFA and FFA are matched through careful load-balancing. Shown here: A send buffer decouples NFA and FFA solver for send operations, prevents idle times, and allows for a more flexible processor assignment

we show stand-alone upscaling results. Other aspects are considered elsewhere: (a) the mapping accuracy is analyzed in Sect. 5, (b) the effectiveness of our load balancing approach as well as the buffering for uni-directional coupling are covered in Sect. 6. If not denoted otherwise, the following measurements are performed on the supercomputing systems SuperMUC[5] and HazelHen.[6]

Mapping Initialization: Preallocation and Matrix Filling As described previously, one of the key components of mapping initialization is the spatial tree which allows for performance improvements by accelerating the interpolation matrix construction. Figure 3 compares different approaches to matrix filling and preallocation: (a) no preallocation: using no preallocation at all, i.e., allocating each entry separately, (b) explicitly computed: calculate matrix sparsity pattern in a first mesh traversal, allocate entries afterwards, and finally fill the matrix in a second mesh traversal, (c) computed and saved: additionally cache mesh element/data point relations from the first mesh traversal and use them in the second traversal to fill the matrix with less computation, (d) spatial tree: use the spatial tree instead of brute-force pairwaise comparisons to determine mesh components relevant for the

[5] 28× Intel-Xeon-E5-2697 cores, 64 GB memory per node.
[6] 24× Intel Xeon-E5-2680 cores, 128 GB memory per node.

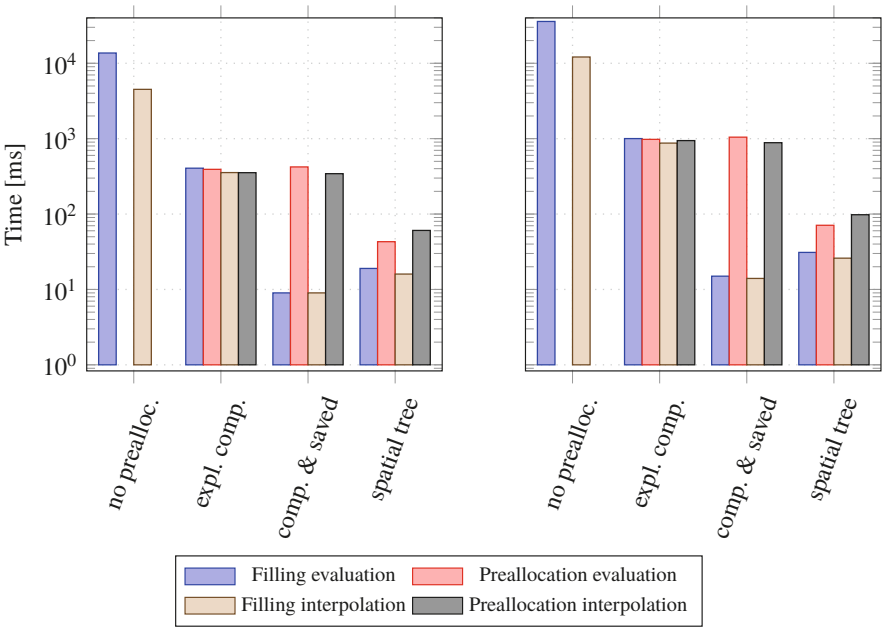


Fig. 3 Mapping initialization. Comparison of different preallocations methods for mesh sizes 6400 (left sub-figure) and 10,000 (right sub-figure) on two ranks per participant. The plot compares times spent in the stages of preallocation and filling of matrices for both the evaluation matrix and the interpolation matrix of an RBF mapping with localized Gaussian basis functions including 6 vertices of the mesh. The total time required is the sum of all bars of one measurement. Note the logarithmic scaling of the y-axis. The measurements were performed on one node of the `sgscl1` cluster, using 4× Intel Xeon E3-1585 CPUs

mapping. Each method can be considered as an enhancement of the previous one. As it becomes obvious from Fig. 3, the spatial tree was able to provide us a with an acceleration of more than two orders of magnitude.

Communication For communication and its initialization, we only present results for the new single-communicator MPI based solution. For TCP socket communication that still requires the exchange of many connection tokens by means of the file system, we only give a rough factor of 2.5 that we observed in terms of acceleration of communication initialization. Note that this factor can be potentially higher as the number of processes and, thus, connections grows larger, and that the hash-based approach removed the hard limit of ranks per participant inherent to the old approach.

In Figs. 4, 5 and 6, we compare performance results for establishing an MPI connection among different ranks using many-communicators for 1-to-1 connections with using a single communicator representing an n-to-m connection. In our academic setting, both Artificial Solver Testing Environment (ASTE) participants run on n cores. On SuperMUC, each rank connects to $0.4n$ ranks, on HazelHen, with

www.dbooks.org

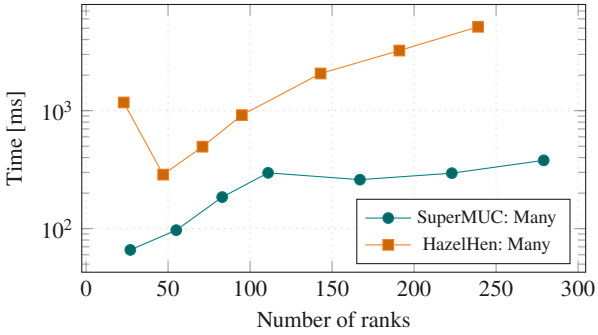


Fig. 4 Communication. Publishing of MPI connection information from participant A for the many-communicator approach. The timings of the new single-communicator approach are not shown, as they are almost negligible with a maximum of 2 ms

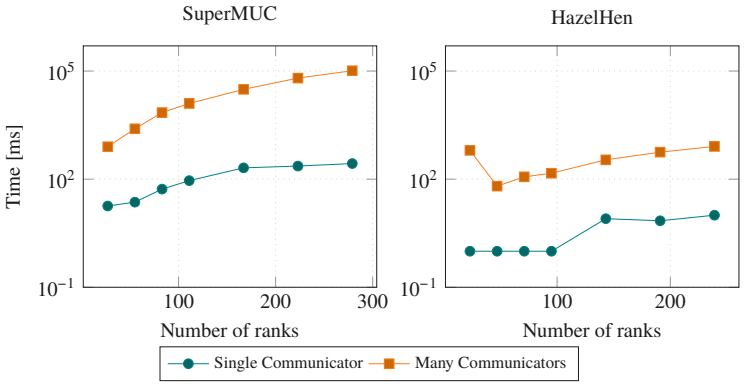


Fig. 5 Communication. Runtime for establishing the connection between the participants using `MPI_Comm_accept` and `MPI_Comm_connect`

a higher number of ranks per node, each rank connects to $0.3n$ ranks. The amount of data transferred between each connected pair of ranks is held constant with 1000 rounds of transfer of an array of 500, and 4000 double values from participant B to participant A. Each measurement is performed five times of which the fastest and the slowest runs are ignored and the remaining three are averaged. We present timings from rank zero, which is synchronized with all other ranks by a barrier, making the measurements from each rank identical. Note, that the measurements are not directly comparable between SuperMUC and HazelHen due to the different number of cores per node and that the test case is even more challenging than actual coupled simulations. In an actual simulation, the number of partner ranks per rank of a participant is constant with increasing number of cores on both sides.

Figure 4 shows the time to publish the connection token. The old approach requires to publish many tokens, which obviously becomes a performance bottleneck as the simulation setup moves to higher number of ranks. The new approach,

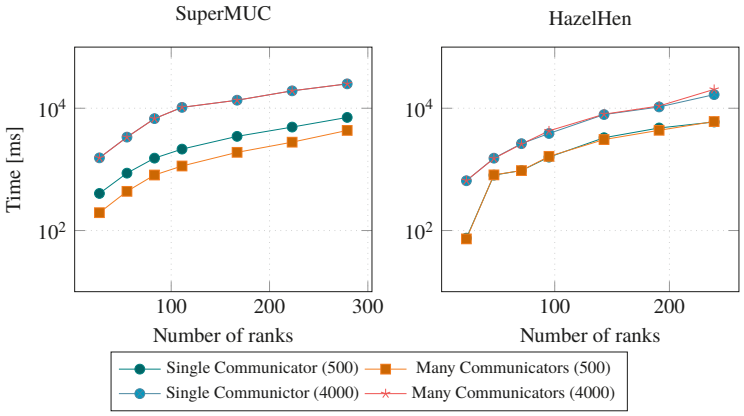


Fig. 6 Communication. Times for 1000 rounds of data transfer of a vector of 500 or 4000 doubles, respectively, from participant B to A. For the transfer, the synchronous MPI routines (`MPI_Send` and `MPI_Recv`) have been used

on the other hand, only publishes one token. It is omitted in the plot, as the times are negligible (<2 ms). In Fig. 5, the time for the actual creation of the communicator is presented. The total number of communication partners per communicator is smaller with the old many-communicator concept (as the communication topology is sparse). However, the creation of many 1-to-1 communicators is substantially slower than the creation of one all-to-all communicator for both HPC systems. Finally, in Fig. 6 the performance for an exchange of data sets of two different sizes is presented. The results for single- and many-communicator approaches are mostly on par with the notable exception of the SuperMUC system. There, the new approach suffers a small but systematic slow-down for small message sizes. We argue that this is a result of vendor specific settings of the MPI implementation.

Data Mapping As described above, we have further improved the mapping initialization, in particular by applying a tree-based approach to identify data dependencies induced by the mapping between grid points of the non-matching solver grids and to assemble the interpolation matrix for RBF mapping. Accordingly, we show both the reduction of the matrix assembly runtime (Fig. 3) and the scalability of the mapping, including setting up the interpolation system and the communication initialization.

These performance tests of preCICE are measured using a special testing application called ASTE.[7] This application behaves like a solver to preCICE but provides artificial data. It is used to quickly generate input data and decompose it for upscaling tests. ASTE generates uniform, rectangular, two-dimensional meshes on $[0, 1] \times [0, 1]$ embedded in three-dimensional space with the z-dimension always

[7] https://github.com/precice/aste.

www.dbooks.org

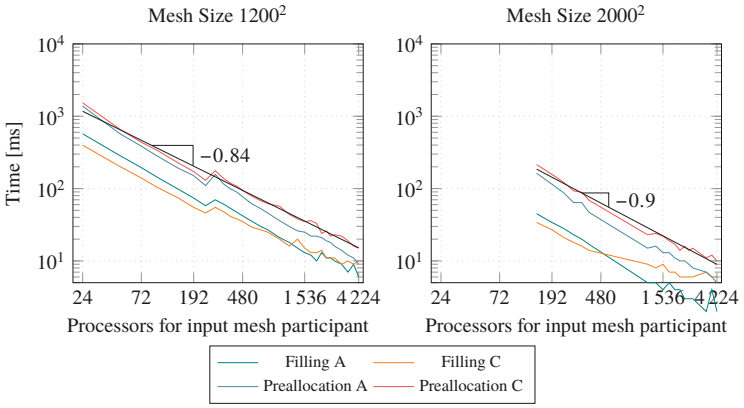


Fig. 7 Data mapping initialization. Strong scaling of the initialization of the interpolation matrix C and the evaluation matrix A for RBF interpolation on mesh sizes 1200^2 and 2000^2 with Gaussian ($m = 6$) basis functions on up to 4224 processors on the HazelHen HPC system

set to zero. The mesh is then decomposed using a *uniform* approach, thus producing partitions of same size as far as possible. Since we mainly look at the mapping part which is only executed as one of the participants, we limit the upscaling to this participant. The other participant always uses one node (28 resp. 24 processors). The mesh size is kept constant, i.e., we perform a strong scaling. The upscaling of an RBF mapping with Gaussian basis functions is shown in Fig. 7.

5 Black-Box Coupling Versus White-Box Coupling with APESMate

In the above section, we have evaluated the performance of the black-box coupling tool preCICE. In this section, we introduce an alternative approach that allows to couple different solvers provided within the framework APES [31]. Black-box data mapping in preCICE only requires point values (nearest neighbor and RBF mapping), and in some cases (nearest projection) connectivity information on the coupling interface. The white-box coupling approach of APESmate [25] has knowledge about the numerical schemes within the domain, since it is integrated in the APES suite, and has access to the common data-structure TreELM [22]. APESmate can directly evaluate the high order polynomials of the underlying Discontinuous Galerkin scheme. Thus, the mapping in preCICE is more generally applicable, while the approach in APESmate is more efficient in the context of high order scheme. Furthermore, APESmate allows the coupling of all solvers of

the APES framework, both in terms of surface and in terms of volume coupling. The communication between solvers can be done in a straightforward way as all coupling participants can be compiled as modules into one single application. Each subdomain defines its own MPI sub-communicator, a global communicator is used for the communication between the subdomains. During the initialization process, coupling requests are locally gathered from all subdomains and exchanged in a round-robin fashion. As all solvers in APES are based on an octree data-structure and a space-filling curve for partitioning, it is rather easy to get information about the location of each coupling point on the involved MPI ranks. In the following, we compare both accuracy and runtime of the two coupling approaches for a simple academic test case that allows to control the 'difficulty' of the mapping by adjusting order and resolution of the two participants.

Test Case Setup We consider the spreading of a Gaussian pressure pulse over a cubic domain of size $5 \times 5 \times 5$ unit length, with an ambient pressure of 100,000 Pa and a density of 1.0 kg/m^3. The velocity vector is set to 0.0 for all spatial directions. To generate a reference solution, this test case is computed monolithically using the inviscid Euler equations.

For the coupled simulations, we decompose the monolithic test case domain into an inner and an outer domain. The resolution and the discretization order of the inner domain are kept unchanged. In the outer domain, we choose the resolution and the order such that the error is balanced with that of the inner domain. See [15] for the respective convergence study. To be able to determine the mapping error at the coupling interface between inner and outer domain, we choose the time horizon such that the pressure pulse reaches the outer domain, but is still away from the outer boundaries to avoid any influences from the outer boundaries. The test case is chosen in a way, that the differences between the meshes at the coupling interface increase, thus increasing the difficulty to maintain the overall accuracy in a black-box coupling approach. Table 1 provides an overview of all combinations of resolution and order in the outer domain used for our numerical experiments, where the total number of elements per subdomain is given as nElements, the number of coupling points with nCoupling points and the scheme order by nScheme order, respectively. For time discretization, we consider the explicit two stage Runge-Kutta scheme with a time step size of 10^{-6} for all simulations.

Table 1 White-box coupling test scenario with Gaussian pressure pulse combinations of orders and resolution used for the evaluation of the mapping methods

	Test case a		Test case b		Test case c		Test case d	
	Inner	Outer	Inner	Outer	Inner	Outer	Inner	Outer
nElements	32,768	124,000	32,768	7936	32,768	992	32,768	124
nCoupling points	55,296	9600	55,296	3456	55,296	1536	55,296	1176
nScheme order	3	4	3	6	3	8	3	14

www.dbooks.org

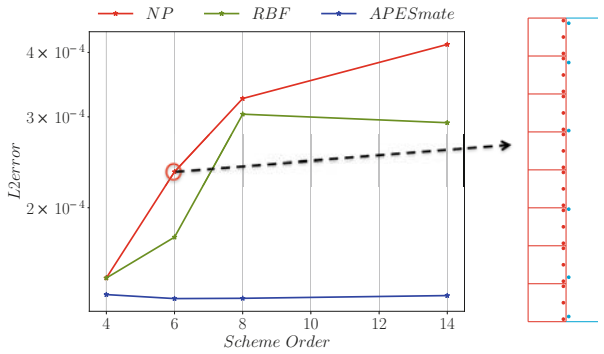


Fig. 8 Mapping accuracy black-box (APESmate) versus white-box (preCICE) approach. Comparison of the L2 error (with the analytical solution as reference) for the Gaussian pressure pulse test case variants and exemplary illustration of the coupling point distribution when using DG for test case (right figure). We compare the black-box data mapping methods Radial-Basis Functions (RBF) interpolation and Nearest Projection (NP) with the direct white-box evaluation of APESmate. The RBF mapping uses local Gaussian basis functions covering three mesh points in every direction

Mapping Accuracy In terms of mapping accuracy, it is expected, that the APESmate coupling is order-preserving, and by that not (much) affected by the increasing differences between the non-matching grids at the coupling interface, while preCICE should show an increasing accuracy drop when the points become less and less matching. This is the case for increasing order of the discretization in the outer domain. Figure 8 illustrates first results. As can be clearly seen, the white-box coupling approach APESmate provides outstanding results by maintaining the overall accuracy of the monolithic solution for all different variations of the coupled simulations, independent of the degree to which the grids are non-matching (increasing with increasing order used in the outer domain). For the interpolation methods provided by preCICE, the error increases considerably with increasing differences between the grids at the interface. As the error of the interpolation methods depends on the distances of the points (see Fig. 8), the error is dominated by the large distance of the integration points in the middle of the surface of an octree grid cell in the High Order Discontinuous Galerkin discretization.

Accuracy Improvement by Regular Subsampling We can decrease the L2 error of NP and RBF and improve the solution of the coupled simulation by providing values at equidistant points on the Ateles side as interpolation support points. The number of equidistant points is equal to the number of coupling points, hence as high as the scheme order. With this new implementation, the error shown in Fig. 9a decreases considerably compared to the results in Fig. 8. We achieve an acceptable accuracy for all discretization order combinations. However, the regular subsampling of values in the Ateles solver increases the overall computational time substantially as can be seen in Fig. 9b.

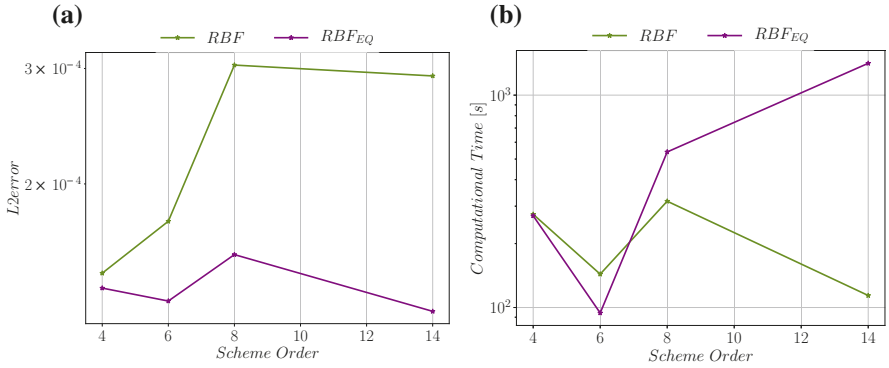


Fig. 9 Mapping accuracy and runtime of black-box with equidistant subsampling versus white-box approach. L2 error behavior (**a**) and computational time (**b**) for the RBF interpolation, when using equidistant (RBF_{EQ}) and non-equidistant (RBF) point distributions for data mapping

To improve the NP interpolation, it turned out that in addition to providing equidistant points, oversampling was required to increase the accuracy. Our investigation showed, that an oversampling factor of 3 is needed to achieve almost the same accuracy as APESmate. In spite of the additional cost of many newly generated support points, the runtime does not increase as much as for RBF, since for the RBF a linear equation system has to be computed, while for NP a simple projection needs to be done.

Summary and Runtime Comparison Figure 10 shows a summary of all tested methods for the interpolation/evaluation before and after improvements. The integrated coupling approach APESmate provides not just very accurate results, but also low runtimes. At this point, we want to recall that this is as expected—the white-box approach makes use of all internal knowledge, which gives it advantages in

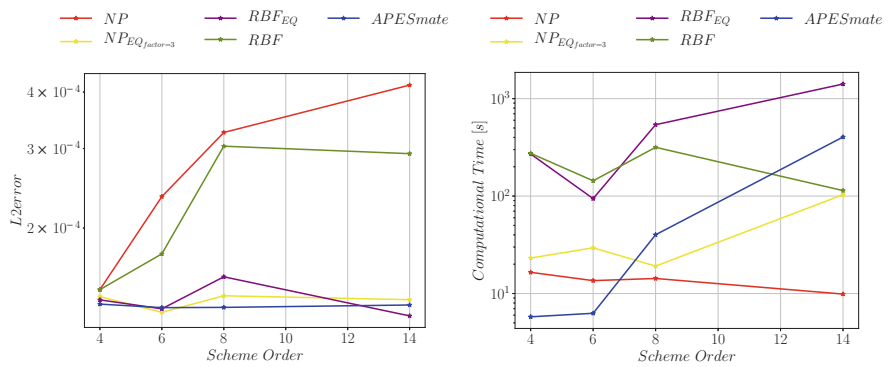


Fig. 10 Mapping accuracy and runtime summary of black-box versus white-box approach. L2 error and computational time for all methods

www.dbooks.org

terms of accuracy and efficiency. On the other hand, this internal knowledge binds it to the solvers available in the framework, while preCICE can be applied to almost all available solvers. Further details regarding this investigation can be found in [15, 16].

6 Results

This section presents a more realistic test case, the turbulent flow over a fence, to assess the overall performance of our approach. Analyses for accuracy and specific isolated aspects are integrated in the sections above.

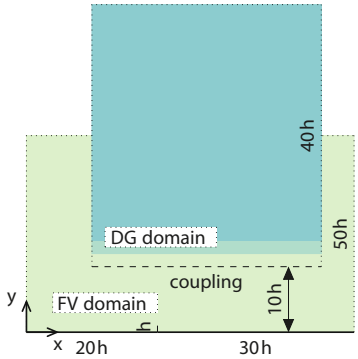
6.1 Flow over a Fence Test Case Setup

As a test case to assess the overall scalability, we simulate the turbulent flow over a (flexible) fence and the induced acoustic far-field as already shown schematically in Fig. 1. The FSI functionality of FASTEST has been demonstrated earlier many times, e.g. in [34]. Thus we focus on the acoustic coupling.

As boundary conditions, we use a no-slip wall at the bottom and the fence surface, an inflow on the left with u_{bulk}, outflow convective boundary conditions on the right, periodic boundary conditions in y-directions, and slip conditions at the upper boundary for the near-field flow. For the acoustic perturbation, we apply reflection conditions at the bottom and the fence surface, zero-gradient condition at all other boundaries. The acoustic far-field solver uses Dirichlet boundary conditions at its lower boundary (see also Eq. (9)). Therefore, the upper near-field boundary is not the coupling interface, but we instead overlap near-field and far-field as shown in Fig. 11. Figures 12 and 13 show a snapshot of the near-field flow and the near-field and far-field acoustic pressure, respectively.

6.2 Fluid-Acoustics Coupling with FASTEST and Ateles

To demonstrate the computational performance of our framework using FASTEST for the flow simulation in the near-field, the high-order DG solver Ateles for the far-field acoustic wave propagation, and preCICE for coupling, we show weak scalability measurements for the interaction between near-field flow simulation and far-field acoustics. We keep both the mesh and the number of MPI ranks in the near-field flow simulation fixed. In the far-field computed with Ateles, we refine the mesh to better capture the acoustic wave propagation. We use a multi-level mesh with a fine mesh at the coupling interface to allow a smooth solution at the coupling interface between the near-field and the far-field. We refine the far-field mesh in



Fence height	$h = 0.05m$
Inflow velocity	$u_{\text{bulk}} = 50m/s$
Fluid material	air
Fluid density	$\rho = 1.184kg/m^3$
Dynamic viscosity	$\mu = 1.586 \cdot 10^{-5} kg/(ms)$
Speed of sound	$c = 343.3m/s$
Young's modulus	$E = 16MPa$
Poisson ratio	$\nu = 0.48$
Structure density	$\rho_S = 1360kg/m^3$

Fig. 11 Flow over a fence test case. Schematic view of the computational domain (left) and applied parameters (right). Colors indicate spatial discretization order in the various regions: The FV domain is completely second order, while the DG domain has second order at the coupling interface for reduced coupling interpolation errors, and subsequently increases the order in various layers for the far-field transport

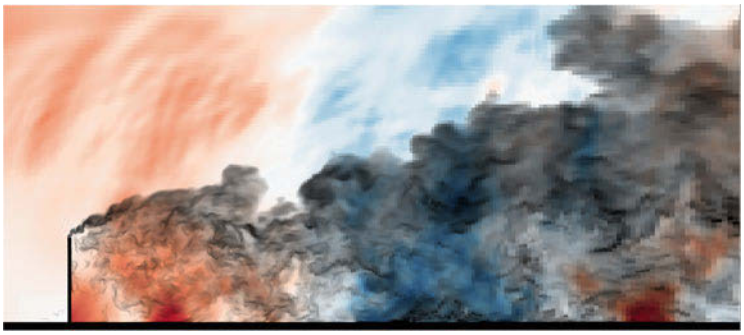


Fig. 12 Flow over a fence test case with FASTEST. Snapshot of the flow in the recirculation area behind the fence. Red/blue indicate acoustic pressure, grey shades show the modelled turbulent kinetic energy (for a $\zeta - f$ DES model)

two main steps: in the first step, we only refine the mesh at the coupling interface. In the next step, we first refine the whole mesh, and again the mesh at the coupling interface in the third and fourth step. Due to the refinement at the coupling interface, the number of Ateles ranks participating in the interface increases such that this study also shows that the preCICE communication does not deteriorate scalability. Table 2 gives an overview of the configurations used for the weak scaling study.

To find the optimal core distribution for all setups, the load balancing approach proposed in Sect. 4 is used. This analysis shows that for the smallest mesh resolution with 24,864 elements in the far-field, the optimal core distribution is 424 cores for the near-field domain and 196 cores for the far-field. For all other setups, we assume perfect scalability, i.e., we choose the number of cores proportional to the number of degrees of freedom in the weak scaling study and increase the number of cores

www.dbooks.org

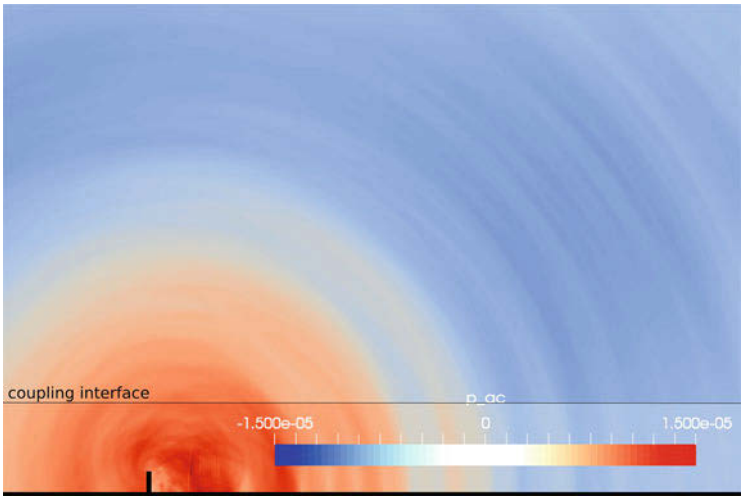


Fig. 13 Flow over a fence test case. Snapshot of the acoustic pressure in a coupled simplified setup

Table 2 Flow over a fence test case

Cores in FASTEST/Ateles	Ateles degrees of freedom	Ateles elements
424/196	16,116,480	24,864
424/756	62,535,840	89,376
424/1428	116,524,800	177,408
424/3136	254,150,400	607,488
424/15,372	1,245,054,720	3,704,064

Scalability study for the interaction between the near-field flow simulation and the far-field acoustics: Summary of mesh details and core numbers for weak scaling. In the FASTEST simulation of the near-field flow simulation, we use 52,822,016 elements. In the far-field, Ateles uses discretization order 9

simultaneously by a factor of two in both fields for strong scaling. The scalability measurements are shown in Fig. 14. The results show that the framework scales almost perfectly up to 6528 cores.

6.3 Fluid-Acoustics Coupling with Only Ateles

In Sect. 5 we investigated the suitability of different interpolation methods for our simulations. In this section, we present a strong scaling study for an Ateles-Ateles coupled simulation of the flow over a fence test case. The fence is modelled in

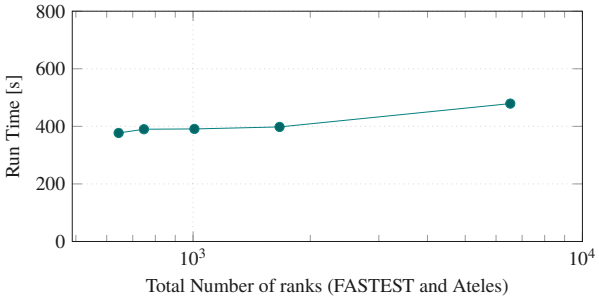


Fig. 14 Flow over a fence test case. Weak scalability measurement of the fluid-acoustic interaction simulation for the fence test case

Ateles using the newly implemented immersed boundary method, enabling high-order representation of complex geometries in Ateles [1]. We solve the compressible Navier-Stokes equations in the flow domain with a scheme order of 4 and a four step mixed implicit-explicit Runge-Kutta time stepping scheme, with a time step size of 10^{-7}. The total number of elements in the flow domain is 192,000. For the far-field, we use the same setup as for the FASTEST-Ateles coupling.

The linearized Euler equations in the far-field can be solved in a DG setting in the modal formulation, which makes the solver very cheap even for very high order. In the near-field domain, the non-linear Navier-Stokes equations are solved with a more expensive hybrid nodal-modal approach. Due to this, and the different spatial discretizations and scheme orders, both domains have different computational load, which requires load balancing. We use static load balancing, since neither the mesh nor the scheme order vary during runtime. As both solvers are instances of Ateles, we apply the SpartA algorithm [21], which allows re-partitioning of the workload according to weights per elements, which are computed during runtime. Those weights are then used to re-distribute the elements according to the workload among available processes (see [17] for more details). The total number of processes used for this test case are 14,336 processes, which is equal to one island on the system. As mentioned previously, the total workload per subdomain does not change, therefore we start our measurements by providing the lower subdomain 100 processes and the upper subdomain 12 processes, which is equal to 4 nodes on the system. This number per subdomain is then doubled for each run, the ratio is kept the same.

Figure 15 shows the strong scaling measurements for both coupling approaches (APESmate and preCICE) executed on the SuperMUC Phase2 system. As can be clearly seen, both coupling setups Ateles-APESmate-Ateles and Ateles-preCICE-Ateles scale almost ideally, however with a lower absolute runtime for the APESmate coupling as expected.

www.dbooks.org

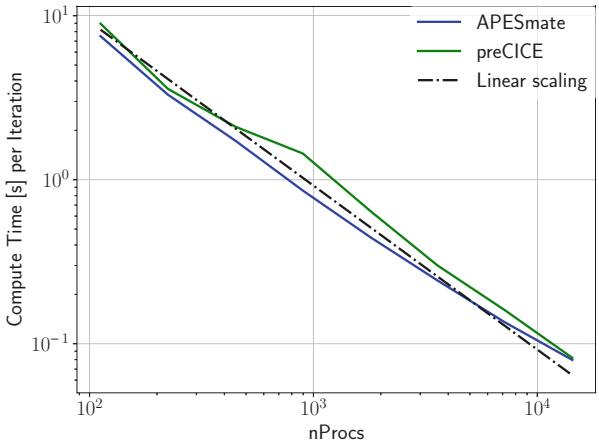


Fig. 15 Flow over a fence test case. Strong scaling measurement for Ateles-Ateles coupling using both APESmate and preCICE—and the ideal linear scaling as reference

7 Summary and Conclusion

We have presented a partitioned simulation environment for the massively parallel simulation of fluid-structure-acoustic interactions. Our setup uses the flow and acoustic solvers in the finite volume software FASTEST, the acoustic solvers in the discontinuous Galerkin framework Ateles as well as the black-box fully parallel coupling library preCICE. In particular, we could show that with a careful design of the coupling tool as well as of solver details, we can achieve a bottleneck-free numerically and technically highly scalable solution. It turned out that efficient initialization of point-to-point communication relations and mapping matrices between the involved participants, sophisticated inter-code load balancing and asynchronous communication using message buffering are crucial for large-scale scenarios. With these improvements, we advanced the limits of scalability of partitioned multi-physics simulations from less than a hundred cores to more than 10,000 cores. Beyond that, we reach a problem size that is not required by the given problem as well as scalability limits of the solvers. The coupling itself is not the limiting factor for the given problem size and degree of parallelism. To be able to use also vector architectures in an efficient sustainable way, we adapted our solvers with a highly effective code transformation approach.

Acknowledgments We thankfully acknowledge funding from SPPEXA, the German Science Foundation Priority Program 1648—Software for Exascale Computing and the European Union's Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 754462, and compute time granted on the LRZ SuperMUC system and the HLRS HazelHen system.

References

1. Anand, N., Pour, N.E., Klimach, H., Roller, S.: Utilization of the brinkman penalization to represent geometries in a high-order discontinuous Galerkin scheme on octree meshes. Symmetry **11**(9), 1126 (2019). https://doi.org/10.3390/sym11091126
2. Banks, J.W., Henshaw, W.D., Schwendeman, D.W.: An analysis of a new stable partitioned algorithm for FSI problems. Part I: incompressible flow and elastic solids. J. Comput. Phys. **269**, 108–137 (2014)
3. Bazilevs, Y., Takizawa, K., Tezduyar, T.E.: Computational Fluid-Structure Interaction: Methods and Applications. Wiley, Hoboken (2012)
4. Blom, D., Ertl, T., Fernandes, O., Frey, S., Klimach, H., Krupp, V., Mehl, M., Roller, S., Sternel, D.C., Uekermann, B., Winter, T., van Zuijlen, A.: Partitioned fluid-structure-acoustics interaction on distributed data – numerical results and visualization. In: Bungartz, H.J., Neumann, P., Nagel, E.W. (eds.) Software for Exa-scale Computing – SPPEXA 2013–2015. Springer, Berlin (2016)
5. Boer, A.D., van Zuijlen, A., Bijl, H.: Comparison of conservative and consistent approaches for the coupling of non-matching meshes. Comput. Methods Appl. Mech. Eng. **197**(49), 4284–4297 (2008)
6. Bungartz, H., Mehl, M., Schäfer, M.: Fluid Structure Interaction II: Modelling, Simulation, Optimization. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2010)
7. Bungartz, H.J., Lindner, F., Mehl, M., Uekermann, B.: A plug-and-play coupling approach for parallel multi-field simulations. Comput. Mech. **55**(6), 1119–1129 (2015)
8. Bungartz, H.J., Lindner, F., Gatzhammer, B., Mehl, M., Scheufele, K., Shukaev, A., Uekermann, B.: preCICE – a fully parallel library for multi-physics surface coupling. Comput. Fluids **141**, 250–258 (2016). Advances in Fluid-Structure Interaction
9. Bungartz, H.J., Lindner, F., Mehl, M., Scheufele, K., Shukaev, A., Uekermann, B.: Partitioned fluid-structure-acoustics interaction on distributed data – coupling via preCICE. In: Bungartz, H.J., Neumann, P., Nagel, E.W. (eds.) Software for Exa-scale Computing – SPPEXA 2013–2015. Springer, Berlin (2016)
10. Calotoiu, A., Beckinsale, D., Earl, C.W., Hoefler, T., Karlin, I., Schulz, M., Wolf, F.: Fast multi-parameter performance modeling. In: 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp. 172–181 (2016). https://doi.org/10.1109/CLUSTER.2016.57
11. Caretto, L.S., Gosman, A.D., Patankar, S.V., Spalding, D.B.: Two calculation procedures for steady, three-dimensional flows with recirculation. In: Proceedings of the Third International Conference on Numerical Methods in Fluid Dynamics. Springer, Paris (1972)
12. Crosetto, P., Deparis, S., Fourestey, G., Quarteroni, A.: Parallel algorithms for fluid-structure interaction problems in haemodynamics. SIAM J. Sci. Comput. **33**(4), 1598–1622 (2011)
13. Degroote, J., Bathe, K.J., Vierendeels, J.: Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. Comput. Struct. **87**(11–12), 793–801 (2009)
14. Dhondt, G.: The Finite Element Method for Three-Dimensional Thermomechanical Applications. Wiley, Chichester (2004)
15. Ebrahimi Pour, N., Roller, S.: Error investigation for coupled simulations using discontinuous galerkin method for discretisation. In: Proceedings of ECCM VI/ECFD VII, Glasgow, June 2018
16. Ebrahimi Pour, N., Krupp, V., Klimach, H., Roller, S.: Coupled simulation with two coupling approaches on parallel systems. In: Resch, M.M., Bez, W., Focht, E., Gienger, M., Kobayashi, H. (eds.) Sustained Simulation Performance 2017, pp. 151–164. Springer International Publishing (2017). https://doi.org/10.1007/978-3-319-66896-3_10
17. Ebrahimi Pour, N., Krupp, V., Roller, S.: Load balancing for coupled simulations. In: Resch, M.M., Bez, W., Focht, E., Gienger, M., Kobayashi, H. (eds.) Sustained Simulation Performance 2019. Springer International Publishing, Cham (2019)

www.dbooks.org

18. Fachgebiet für Numerische Berechnungsverfahren im Maschinenbau: FASTEST Manual (2005)
19. Gee, M.W., Küttler, U., Wall, W.A.: Truly monolithic algebraic multigrid for fluid–structure interaction. Int. J. Numer. Methods Eng. **85**(8), 987–1016 (2011)
20. Haelterman, R., Degroote, J., van Heule, D., Vierendeels, J.: The quasi-Newton least squares method: a new and fast secant method analyzed for linear systems. SIAM J. Numer. Anal. **47**(3), 2347–2368 (2009)
21. Harlacher, D.F., Klimach, H., Roller, S., Siebert, C., Wolf, F.: Dynamic load balancing for unstructured meshes on space-filling curves. In: 2012 IEEE 26th International on Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), pp. 1661–1669 (2012)
22. Klimach, H.G., Hasert, M., Zudrop, J., Roller, S.P.: Distributed octree mesh infrastructure for flow simulations. In: Eberhardsteiner, J. (ed.) ECCOMAS 2012 - European Congress on Computational Methods in Applied Sciences and Engineering, e-Book Full Papers (2012)
23. Kong, F., Cai, X.C.: Scalability study of an implicit solver for coupled fluid-structure interaction problems on unstructured meshes in 3D. Int. J. High Perform. Comput. Appl. **32**, 207–219 (2016)
24. Kornhaas, M., Schäfer, M., Sternel, D.: Efficient numerical simulation of aeroacoustics for low Mach number flows interacting with structures. Comput. Mech. **55**, 1143–1154 (2015). https://doi.org/10.1007/s00466-014-1114-1
25. Krupp, V., Masilamani, K., Klimach, H., Roller, S.: Efficient coupling of fluid and acoustic interaction on massive parallel systems. In: Sustained Simulation Performance 2016, pp. 61–81 (2016)
26. Lindner, F.: Data transfer in partitioned multi-physics simulations: interpolation & communication. PhD thesis, University of Stuttgart (2019)
27. Lindner, F., Mehl, M., Uekermann, B.: Radial basis function interpolation for black-box multi-physics simulations. In: Papadrakakis, M., Schrefler, B., Onate, E. (eds.) VII International Conference on Computational Methods for Coupled Problems in Science and Engineering, pp. 1–12 (2017)
28. Link, G., Kaltenbacher, M., Breuer, M., Döllinger, M.: A 2D finite-element scheme for fluid-solid-acoustic interactions and its application to human phonation. Comput. Methods Appl. Mech. Eng. **198**(41–44), 3321–3334 (2009)
29. Mehl, M., Uekermann, B., Bijl, H., Blom, D., Gatzhammer, B., van Zuijlen, A.: Parallel coupling numerics for partitioned fluid-structure interaction simulations. Comput. Math. Appl. **71**(4), 869–891 (2016)
30. Reimann, T.: Numerische simulation von fluid-struktur-interaktion in turbulenten strömungen. Ph.D. thesis, Technische Universität Darmstadt (2013)
31. Roller, S., Bernsdorf, J., Klimach, H., Hasert, M., Harlacher, D., Cakircali, M., Zimny, S., Masilamani, K., Didinger, L., Zudrop, J.: An adaptable simulation framework based on a linearized octree. In: Resch, M., Wang, X., Bez, W., Focht, E., Kobayashi, H., Roller, S. (eds.) High Performance Computing on Vector Systems 2011, pp. 93–105. Springer, Berlin (2012)
32. Ross, M.R., Felippa, C.A., Park, K., Sprague, M.a.: Treatment of acoustic fluid-structure interaction by localized Lagrange multipliers: formulation. Comput. Methods Appl. Mech. Eng. **197**(33–40), 3057–3079 (2008)
33. Schäfer, F., Müller, S., Uffinger, T., Becker, S., Grabinger, J., Kaltenbacher, M.: Fluid-structure-acoustic interaction of the flow past a thin flexible structure. AIAA J. **48**(4), 738–748 (2010)
34. Schäfer, M., Sternel, D.C., Becker, G., Pironkov, P.: Efficient numerical simulation and optimization of fluid-structure interaction. In: Bungartz, H.J., Mehl, M., Schäfer, M. (eds.) Fluid Structure Interaction II. Modelling, Simulation, Optimization. Lecture Notes in Computational Science and Engineering, vol. 73, pp. 131–158. Springer, Berlin (2010)
35. Scheufele, K., Mehl, M.: Robust multisecant quasi-Newton variants for parallel fluid-structure simulations – and other multiphysics applications. SIAM J. Sci. Comput. **39**(5), 404–433 (2017)
36. Stone, H.L.: Iterative solution of implicit approximations of multidimensional partial differential equations. SIAM J. Numer. Anal. **5**(3), 530–558 (1968)

37. Totounferoush, A., Ebrahimi Pour, N., Schroder, J., Roller, S., Mehl, M.: A new load balancing approach for coupled multi-physics simulations (2019). https://doi.org/10.1109/IPDPSW.2019.00115
38. Turek, S., Hron, J., Madlik, M., Razzaq, M., Wobker, H., Acker, J.: Numerical simulation and benchmarking of a monolithic multigrid solver for fluid-structure interaction problems with application to hemodynamics. In: Bungartz, H.J., Mehl, M., Schäfer, M. (eds.) Fluid Structure Interaction II: Modelling, Simulation, Optimization, p. 432. Springer, Berlin (2010)
39. Uekermann, B.: Partitioned fluid-structure interaction on massively parallel systems. Ph.D. thesis, Department of Informatics, Technical University of Munich (2019)
40. Uekermann, B., Bungartz, H.J., Cheung Yau, L., Chourdakis, G., Rusch, A.: Official preCICE adapters for standard open-source solvers. In: 7th GACM Colloquium on Computational Mechanics, pp. 1–4 (2017)
41. Vierendeels, J., Lanoye, L., Degroote, J., Verdonck, P.: Implicit coupling of partitioned fluid-structure interaction problems with reduced order models. Comput. Struct. **85**(11–14), 970–976 (2007)
42. Zudrop, J., Klimach, H., Hasert, M., Masilamani, K., Roller, S.: A fully distributed CFD framework for massively parallel systems. In: Cray User Group Conference. Cray User Group, Stuttgart (2012)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

EXAHD: A Massively Parallel Fault Tolerant Sparse Grid Approach for High-Dimensional Turbulent Plasma Simulations



Rafael Lago, Michael Obersteiner, Theresa Pollinger, Johannes Rentrop, Hans-Joachim Bungartz, Tilman Dannert, Michael Griebel, Frank Jenko, and Dirk Pflüger

Abstract Plasma fusion is one of the promising candidates for an emission-free energy source and is heavily investigated with high-resolution numerical simulations. Unfortunately, these simulations suffer from the curse of dimensionality due to the five-plus-one-dimensional nature of the equations. Hence, we propose a sparse grid approach based on the sparse grid combination technique which splits the simulation grid into multiple smaller grids of varying resolution. This enables us to increase the maximum resolution as well as the parallel efficiency of the current solvers. At the same time we introduce fault tolerance within the algorithmic design and increase the resilience of the application code. We base our implementation on a manager-worker approach which computes multiple solver runs in parallel by distributing tasks to different process groups. Our results demonstrate good

R. Lago · T. Dannert
Max Planck Computing and Data Facility, Munich, Germany
e-mail: rafael.lago@mpcdf.mpg.de; tilman.dannert@rzg.mpg.de

M. Obersteiner
Technical University of Munich, Munich, Germany
e-mail: oberstei@in.tum.de

H.-J. Bungartz
Technical University of Munich, Garching, Germany
e-mail: bungartz@in.tum.de

T. Pollinger · D. Pflüger (✉)
University of Stuttgart, Stuttgart, Germany
e-mail: theresa.pollinger@ipvs.uni-stuttgart.de; Dirk.Pflueger@ipvs.uni-stuttgart.de

J. Rentrop · M. Griebel
University of Bonn, Bonn, Germany
e-mail: rentrop@ins.uni-bonn.de; griebel@ins.uni-bonn.de

F. Jenko
Max Planck Institute for Plasma Physics, Munich, Germany
e-mail: Frank.Jenko@ipp.mpg.de

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_11

convergence for linear fusion runs and show high parallel efficiency up to 180k cores. In addition, our framework achieves accurate results with low overhead in faulty environments. Moreover, for nonlinear fusion runs, we show the effectiveness of the combination technique and discuss existing shortcomings that are still under investigation.

1 Introduction

Scientists widely agree that the human-made climate change due to CO_2 emission will pose severe challenges for the future. Hence, to reduce the overall emission of greenhouse gases, carbon free energy sources will be required. Nuclear fusion is one candidate which offers abundant fuel with a large energy output. However, there are still many difficulties to overcome in the task to build an energy-positive fusion reactor. A major challenge are micro-instabilities caused by turbulent plasma flow. These can be studied by numerical simulations to further improve the design of fusion reactors. One of the codes dedicated to tackle this problem is GENE, which solves the gyrokinetic Vlasov-Maxwell equations.

Unfortunately, the computational costs of these simulations are enormous. Since they involve calculations on a five-dimensional spatial grid, they suffer from the curse of dimensionality, i.e. from the exponential dependence of the grid size on dimension. In our project, we overcome this issue with the help of the sparse grid combination technique. This method splits a simulation into several independent runs on coarser anisotropic grids and then aggregates the results on a sparse grid. As a consequence, the curse of dimensionality is alleviated, which makes larger simulations feasible. In addition, the arising subproblems can be computed in parallel which allows for scaling to larger processor numbers. This enables the use of future exascale computers. Finally, the method opens up an algorithmic approach to fault tolerance, which will be a key requirement for exascale computing.

In this paper, we report on the main contributions of the two funding periods of our project EXAHD within the priority program Software for Exascale Computing of the DFG, with an emphasis on the second funding period. Partners in this joint project were the Institute for Parallel and Distributed Systems at the University of Stuttgart (PI Pflüger), the Institute for Numerical Simulations at the University of Bonn (PI Griebel), the Institute for Informatics at the Technical University of Munich (PI Bungartz), the Max-Planck Institute for Plasma Physics (PI Jenko), the Supercomputing Center of the Max-Planck Society Garching (PI Dannert, second period), and the Center for Mathematics and Its Applications of the Australian National University (Hegland, external partner).

Our main contributions include significant progress for the solution of higher-dimensional plasma flow problems. In particular, we addressed several exascale challenges: we have realized the first-ever massively parallel computations with the sparse grid combination technique, enabling scalability beyond the petascale for mesh-based discretizations by numerically decoupling the underlying systems and by introducing a novel level of parallelism. We tackled load-balancing on massively

www.dbooks.org

parallel systems, learning from gathered runtime data. And we have developed new and innovative approaches for fault tolerance on multiple levels, in particular algorithm-based fault tolerance, which can be a crucial aspect in settings where checkpoint-restart is infeasible due to the massive amount of data that would have to be handled.

In the first funding period we have focused on linear simulations of hot fusion plasma which govern the exponential growth phase of turbulent modes. We have developed core algorithms for fault tolerance and scalability, including algorithm-based fault tolerance with the combination technique and optimal communication schemes.

In the second funding period, we have extended the simulation setting to fully nonlinear simulations, which pose several hurdles as they do not match the classical setting for the sparse grid combination technique anymore. We have further developed a flexible framework for massively parallel simulations with the combination technique, including software interfaces to GENE and the general PDE framework DUNE. We have realized fault tolerance within both GENE and the combination framework, which even allows to detect and mitigate soft faults, for example due to silent data corruption. Finally, we have extended load balancing to a fully data-driven approach based on machine learning.

In the following, we first give an introduction to the underlying theory and the mathematical model. To this end, we describe the sparse grid combination technique and the gyrokinetic approach to plasma physics underlying GENE. Then, we outline the principles behind our approach to fault tolerance. The third section focuses on the implementation while numerical results are presented in section four.

2 Theory and Mathematical Model

2.1 The Sparse Grid Combination Technique

The sparse grid combination technique [10] is a method for approximating high-dimensional problems based on sparse grids [2]. It yields an alternative representation of a sparse grid solution, not reliant on hierarchical surpluses but using different coarse full grid solutions to build a combination solution. The underlying idea of the sparse grid approximation was first introduced by Smolyak for the case of quadrature [24] and was since applied to a broad field of applications [7, 21].

Let's assume we want to approximate a given function u. Furthermore let $\Omega_{\mathbf{n}}$ be the regular Cartesian grid on $[0, 1]^d$ with mesh size $\mathbf{h_n} = 2^{-\mathbf{n}} := (2^{-n_1}, \ldots, 2^{-n_d})$ resulting in $2^{n_i} \pm 1$ points along the i-th direction, depending on whether there are points on the left and/or right boundary. Arbitrary rectangular domains can be treated by scaling them onto the unit hypercube.

Now we can define a piecewise linear approximation of u on $\Omega_{\mathbf{n}}$,

$$u_{\mathbf{n}} = \sum_{\mathbf{j}} u_{\mathbf{n},\mathbf{j}} \, \phi_{\mathbf{n},\mathbf{j}} \tag{1}$$

where $u_{\mathbf{n},\mathbf{j}}$ are the function values at the grid points and $\phi_{\mathbf{n},\mathbf{j}}$ are piecewise d-linear hat functions with support of volume $\prod_{i=1}^{d}(2\,h_{n_i})$ anchored at the grid points, which we call *nodal basis*. There are various other types of basis functions one could choose in order to increase the order of the approximation. For various examples we refer to [2]. In this presentation, we stick to the piecewise linear case for reasons of simplicity.

The function Eq. (1) can also be represented in the so called *hierarchical basis*,

$$u_{\mathbf{n}} = \sum_{\boldsymbol{\ell}\le\mathbf{n}} \sum_{\mathbf{j}} \alpha_{\boldsymbol{\ell},\mathbf{j}}\, \hat{\phi}_{\boldsymbol{\ell},\mathbf{j}} := \sum_{\boldsymbol{\ell}\le\mathbf{n}} \hat{u}_{\boldsymbol{\ell}}\ , \tag{2}$$

where the hierarchical basis functions $\hat{\phi}_{\boldsymbol{\ell},\mathbf{j}}$ are those hat functions of level $\boldsymbol{\ell}$ with odd indices only (for $\ell_i > 0$). Here, the expression $\boldsymbol{\ell} \le \mathbf{n}$ is to be understood componentwise. We call the linear transformation between the two representations *hierarchization* and *dehierarchization*, respectively.

Now, if u possesses a certain smoothness property, namely (for our case of piecewise d-linear hierarchical basis functions) that its second mixed derivative is bounded, then the size of the coefficients $\alpha_{\boldsymbol{\ell},\mathbf{j}}$ decays as $\sim 2^{-2|\boldsymbol{\ell}|_1}$. Hence, they are called *hierarchical surplusses*. This leads to the idea of approximating $u_{\mathbf{n}}$ by truncating the sum in Eq. (2). Compared to the full Cartesian grid, merely a subset of points carries a basis function, which is called a *sparse grid*. This can be formalized as follows: For any multi-index set $\mathcal{I}$ that is downward-closed (for any $\boldsymbol{\ell} \in \mathcal{I}$ all $\boldsymbol{\ell}' \le \boldsymbol{\ell}$ are in the set as well) we set

$$u_{\mathbf{n}} = \sum_{\boldsymbol{\ell}\in\mathcal{I}} \hat{u}_{\boldsymbol{\ell}} \tag{3}$$

with $n_i = \max\{\ell_i : \boldsymbol{\ell} \in \mathcal{I}\}$ defining the *target level*. The classical sparse grid found in the literature for an isotropic grid with $\mathbf{n} = n \cdot \mathbf{1}$ is given by $\mathcal{I} = \{\boldsymbol{\ell} \in \mathbb{N}_0^d : |\boldsymbol{\ell}|_1 \le n\}$, i.e. only a standard simplex of levels instead of the full hyperrectangle is taken into account. In this case, as shown in [2], the number of points is drastically reduced, from N^d to $O(N\,(\log N)^{d-1})$, where $N = 2^n \pm 1$. Furthermore, for functions from Sobolev spaces with dominating mixed derivatives H^2_{mix} the asymptotic approximation error with regard to the exact function u is only slightly worse, $O(N^{-2}(\log N)^{d-1})$ compared to the usual $O(N^{-2})$. Generalized sparse grids [4] are defined by means of general downward-closed index sets $\mathcal{I}$ and its associated truncation (3).

Due to the fact that the hierarchical increments $\hat{u}_{\boldsymbol{\ell}}$ can be expressed as differences of full grid functions $u_{\boldsymbol{\ell}}$, Eq. (3) yields a telescopic sum that evaluates to

$$u_{\mathbf{n}}^{(c)} = \sum_{\boldsymbol{\ell}\in\mathcal{I}} c_{\boldsymbol{\ell}}\, u_{\boldsymbol{\ell}} \qquad , \qquad c_{\boldsymbol{\ell}} = \sum_{\mathbf{z}\le\mathbf{1}} (-1)^{|\mathbf{z}|_1}\, \chi_{\mathcal{I}}(\boldsymbol{\ell}+\mathbf{z})\ , \tag{4}$$

www.dbooks.org

with $\chi_{\mathcal{I}}$ being the characteristic function of $\mathcal{I}$. We call this equivalent representation the (generalized) *combination technique*. Note that most of the *combination coefficients* $c_{\boldsymbol{\ell}}$ vanish, except the ones whose upper neighbors are not all included in $\mathcal{I}$ (i.e. the ones near the upper boundary of $\mathcal{I}$). The combination technique is advantageous in practical applications, since code that produces full grid solutions can be readily used as a black box and different *component solutions* $u_{\boldsymbol{\ell}}$ can be computed independently from each other, thus introducing a coarse grain layer of parallelism. The point set produced by the union of all component grids is a sparse grid as before. In practice, if one wants to evaluate the combination solution at a point that is not shared by all component grids, the respective component functions have to be interpolated according to the basis used.

The specific combination scheme used throughout this work is the *truncated planar combination technique*, introducing a minimal level $\boldsymbol{\ell}_{\min}$ and a maximal level $\boldsymbol{\ell}_{\max} \equiv \mathbf{n}$. The formula for the index set reads

$$\mathcal{I}_{\boldsymbol{\ell}_{\min},\boldsymbol{\ell}_{\max}} = \left\{ \boldsymbol{\ell} \in \mathbb{N}_0^d : \boldsymbol{\ell} \in \operatorname{conv}\left(\boldsymbol{\ell}_{\min}, \boldsymbol{\ell}_{\min} + \hat{\boldsymbol{\ell}}^{(1)}, \ldots, \boldsymbol{\ell}_{\min} + \hat{\boldsymbol{\ell}}^{(d)} \right) \right\}, \tag{5}$$

where $\hat{\boldsymbol{\ell}}^{(i)} = (0, \ldots, \ell_{i,\max} - \ell_{i,\min}, \ldots, 0)$, i.e. it includes all level indices that lie inside the simplex spanned by $\boldsymbol{\ell}_{\min}$ and its adjacent corners of the hyperrectangle defined by $\boldsymbol{\ell}_{\min}$ and $\boldsymbol{\ell}_{\max}$. For example, the 2D sparse grid constructed with $\boldsymbol{\ell}_{\min} = (1, 1)$ and $\boldsymbol{\ell}_{\max} = (3, 3)$ is shown in Fig. 1, where the simplex is just a shifted triangle.

As a remark, the approximation rate shown above does not, in general, hold in the case of solving PDEs with the combination technique, which we are interested in. However, it can be shown [3] that the combination technique produces the same rate, as long as a certain ANOVA-like error expansion exists. In fact, the combination technique is applicable to any quantity (not just functions) that is the solution to a problem depending on some discretization parameters, provided there is such an error expansion. Alternatively, there is a second route to proof that the sparse grid Galerkin approximation and the combination method for a PDE possess errors of the same order. To this end, in [8] optimal convergence rates of the combination technique for elliptic operators acting on arbitrary Gelfant triples were shown.

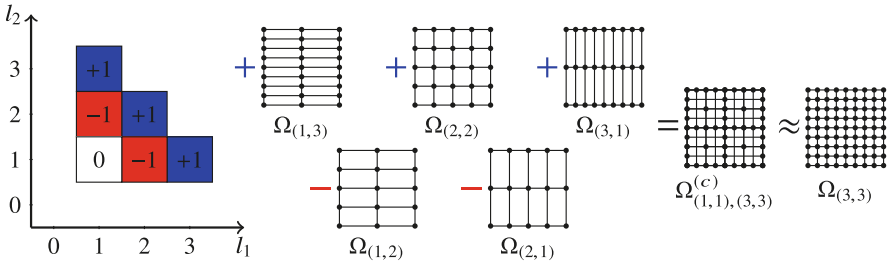


Fig. 1 Grids resulting from a 2D combination technique with $\boldsymbol{\ell}_{\min} = (1, 1)$ and $\boldsymbol{\ell}_{\max} = (3, 3)$. The result of the combination is the sparse grid $\Omega^{(c)}_{(1,1),(3,3)}$

Nevertheless, since it may be too hard to proof for a given case that the necessary error expansion exists, or that the respective Gelfand triple indeed leads to equal approximation rates between the sparse grid Galerkin approach and the combination method, this motivates the use of numerical experiments to determine the applicability of the combination technique.

The type of PDE we will be concerned with is a time-dependent initial value problem. We will, however, not include the time direction as a dimension of the sparse grid approximation, but rather treat it as a parameter. The reason is that the solver we will mainly deal with employs an explicit time stepping method, which means that anisotropic grids with large time steps may be forbidden by a CFL condition. The solver also dynamically adapts the time step size, so that it may be disadvantageous to interfere with its capabilities.

Another difficulty is that the global truncation error introduced by a time stepping scheme usually grows exponentially in time and, for a nonlinear PDE, even the exact solution may be highly sensitive to perturbations in the initial condition (which are always present due to different spatial discretizations). In consequence, we cannot expect the component solutions to stay similar for long simulation times and so the combination technique would break down.

Hence, we will look for the solution after some interval ΔT, starting at time t, then repeat the process beginning at time $t+\Delta T$ and so forth. Using the combination technique for the remaining (spatial) dimensions, the solution is constructed from the time evolution of the component solutions:

$$u_{\mathbf{n}}^{(c)}(t+\Delta T) \coloneqq \sum_{\boldsymbol{\ell}\in\mathcal{I}} c_{\boldsymbol{\ell}}\, \mathcal{T}_{t\to t+\Delta T}\left[u_{\boldsymbol{\ell}}(t)\right] \tag{6}$$

with the time evolution operator $\mathcal{T}$. The initial values are set by interpolating the combination solution from the previous time step onto the different component grids. Once in the beginning they are set by discretizing a known initial function u_0:

$$u_{\boldsymbol{\ell}}(t) \coloneqq I_{\boldsymbol{\ell}}\left[u_{\mathbf{n}}^{(c)}(t)\right] \qquad , \qquad u_{\boldsymbol{\ell}}(0) \coloneqq I_{\boldsymbol{\ell}}\left[u_0\right] \ , \tag{7}$$

where $I_{\boldsymbol{\ell}}$ denotes the interpolation operator onto grid $\Omega_{\boldsymbol{\ell}}$. The interval ΔT has to be chosen sufficiently small, so that the error introduced by the time evolution is small compared to the spatial discretization error (which we want to optimize by using the combination technique). Sensible values for ΔT depend on the scenario at hand and will be investigated in our experiments.

2.2 *Plasma Physics with GENE*

The most common way to study turbulence on a microscopic scale are the gyrokinetic equations. They are a system of partial integro-differential equations for

www.dbooks.org

the particle distribution function f in a 5D phase space plus time for every particle species s in the plasma (typically hydrogen ions and electrons). The model consists of a Vlasov equation (when neglecting collisions) with Lorentz force,

$$\frac{\partial f_s}{\partial t} + \mathbf{v} \cdot \nabla_{\mathbf{x}} f_s + \frac{Z_s e}{m}\left(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B}\right) \cdot \nabla_{\mathbf{v}} f_s = 0 \,, \tag{8}$$

coupled with Maxwell's equations for the electromagnetic fields as well as the self-consistent computation of charge and current density from the respective moments of the distribution function [6].

The gyrokinetic approach uses several approximations tailored to the setting of a suspended plasma in a strong toroidal magnetic field, which is the concept behind tokamaks and stellarators, the most popular types of fusion reactors. Since charged particles move on helical trajectories along magnetic field lines, the motion is split into the direction parallel to the field line, $v_\parallel$, and the plane perpendicular to it. The phase information of the rapid circular motion in the perpendicular direction is then averaged out and its magnitude encoded in the magnetic moment $\mu \propto |v_\perp|^2$. Also, the coordinate system is commonly aligned to the magnetic field lines and, lastly, a δf-splitting is employed, dividing the distribution function into a static equilibrium distribution and an unknown turbulent part g.

Eventually one arrives at a $5 + 1$ dimensional nonlinear partial differential equation for $g(x, y, z, v_\parallel, \mu; t)$ of the general form

$$\frac{\partial g}{\partial t} = \mathcal{L}\, g + \mathcal{N}(g) \tag{9}$$

where $\mathcal{L}$ and $\mathcal{N}$ are the linear resp. nonlinear parts of the differential operator.

One of the most widely used codes to solve the gyrokinetic equations is GENE [17]. It is an Eulerian (fixed grid) solver that uses mostly finite differences to discretize the spatial domain and an explicit Runge-Kutta scheme of fourth order for the time evolution. Even though it is highly optimized and employs domain decomposition to scale well on large computing systems, it still requires huge amounts of computation time and for some scenarios the desired resolution is still out of scope. This is why we deem GENE a good candidate to profit from the combination technique. Not only are the component solutions cheaper in terms of memory, hence always feasible to compute, but additionally can afford a larger time step size (due to the CFL condition), reducing the computational cost further. GENE operates in different modes described here for later reference:

Linear/Nonlinear In the linear mode, $\mathcal{N}(g)$ in Eq. (9) is neglected. In this case one is interested in the growth rate of the turbulence, i.e. the eigenvalue of $\mathcal{L}$ with the largest real part and its corresponding eigenvector. To this end, GENE can either be used as a direct eigenvalue or as an initial value solver. The nonlinear mode treats the full equation and is used to study the quasi-stationary state after the initial growth phase.

Local/Nonlocal The local case uses the so called *flux tube approximation*, simulating only a small cross section of the full torus. Periodic boundary conditions are employed in the x- and y-direction, enabling Fourier transformation. In nonlocal runs the whole domain along the x-direction is treated, losing periodicity in x.

Adiabatic Electrons Setting the number of species to one (only ions), the electrons are not treated with the full kinetic model, which reduces the problem size. This approximation was mostly used for our results.

In order to use GENE within our combination technique framework, several adjustments to GENE had to be implemented during the project. They are minor enough to not undermine the black box functionality. First of all, the grid setup of GENE had to be adjusted so that grids using 2^{ℓ_i} grid points in each respective dimension become nested. This is important because, on the one hand, the combination technique requires nested grids and, on the other hand, powers of two are necessary for an efficient parallelization in GENE. As a result, only the left boundary may carry a grid point (cf. Sect. 2.1), which was originally not the case for every direction. Additionally, the μ-direction uses a Gauss-Laguerre grid by default for efficient quadrature. Here we had to switch to an equidistant grid, losing accuracy, but a potential solution to this problem using Clenshaw-Curtis points is discussed in [18]. Furthermore, GENE requires an unusual quasi-periodic boundary condition in the z-direction, which we had to incorporate into the interface of our framework.

Finally some performance issues had to be overcome. At every restart, GENE performs an initialization routine that produces a large overhead when the simulation time itself is short. In particular for nonlocal runs, a large *gyromatrix* has to be assembled which is costly. To this end we had to integrate the functionality to store and reload this matrix from memory. The same had to be done for the checkpoints which could previously only be written to and read from disk [13].

2.3 Fault Tolerance

2.3.1 Fault Tolerant Combination Technique

The Fault Tolerant Combination Technique (FTCT) is an algorithm-based fault tolerant version of the Sparse Grid Combination Technique, which was proposed in the project proposal of phase one of EXAHD and first published in [12]. The method addresses the problem of both hard faults and soft faults.[1] Since these faults cause a corruption or loss of data on certain processors, it is not guaranteed that all combination grids can safely contribute to the combination solution. After identifying such a fault we therefore need to construct a new combination scheme

[1]Hard faults are detected by the operating system and usually cause the affected system part to fail while soft faults remain undetected by the system and usually appear in the form of bitflips during computation or processing of data.

www.dbooks.org

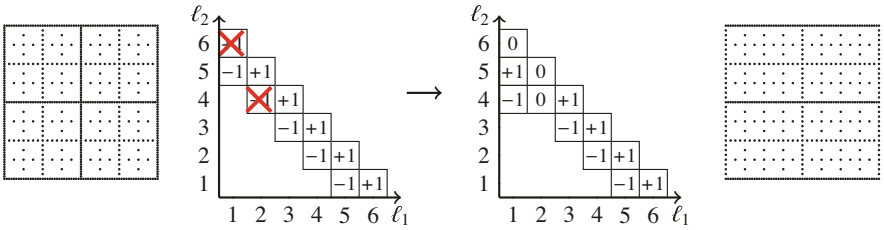


Fig. 2 Recovering the combination scheme after two failed component grids (middle left). A new scheme (middle right) is constructed which excludes the failed resources while utilizing a component grid from the next lower subdiagonal. The resulting sparse grid (right) should be only slightly less accurate than the original sparse grid (left)

that excludes faulty parts from the original scheme. Ultimately, this modified scheme should only consist of existing component grids, thus eliminating the necessity to recompute results. In addition, switching to less and possibly coarser component grids should only slightly decrease the accuracy compared to the original scheme. In case of a time-dependent problem with frequent recombination, the FTCT restores the original combination scheme for all future combinations[2] and continues computation. We will briefly summarize the main steps of the algorithm: fault detection and reconstruction of a fault-free combination scheme.

Fault detection heavily depends on the class of faults that we experience. For hard faults or process failures, typically the operating system in combination with the MPI runtime can be used to identify all processes which are affected by a fault. These faulty ranks are then removed and computation can be continued. In case of soft or silent faults it is more challenging to detect a faulty component as the failure is undetected by the operating system. Therefore, dedicated routines are used to detect such faults, e.g. running a program multiple times or with different algorithms or by checking application dependent properties. An example of the latter case can be found in Sect. 3.3 where we describe our implementation of the silent fault detection.

Once all faulty component grids are detected, the FTCT constructs a new fault-free combination scheme (see Fig. 2). The problem of finding such a new scheme is known as the General Coefficient Problem (GCP). For further information about the problem and on how to solve it most efficiently we refer to [11, 12]. In order to increase convergence speed and the probability of finding such a solution we already compute the component solutions of two additional lower diagonals[3] of the level set from the start, alongside the required ones, even though they originally have a coefficient of zero. Since the degrees of freedom of component grids decrease

[2]In case of process failures either failed resources are replaced by spare ranks or the individual tasks are distributed to the remaining fault-free processes.

[3]For dimensions $d > 2$ it is in fact a $d - 1$ dimensional slice and it might be tilted in general.

exponentially with the magnitude of the level index, this adds only minor overhead to the overall computation time. Section 3.3 outlines our implementation of the FTCT in more detail.

2.3.2 Fault Recovery Algorithms

In addition to the FTCT, a fault tolerant version of GENE was developed (FT-GENE). The purpose was to allow FT-GENE to tackle small faults, whereas larger faults would be handled by the FTCT. To that end, an object oriented Fortran 2008 library called libSpina was written. The purpose of this library is to provide functionalities that aid the implementation of a fault tolerant application. In contrast to existing tools such as ULFM [1], libSpina does not replace any existing MPI implementation and does not attempt to extend the MPI standard.

The library is responsible for managing and separating "spare nodes" from the application, the management of a channel for broadcasting errors, detection of faulty resources and sanitization of the MPI environment. Instead of forcing a complete rewriting of the code in order to fit a fault tolerant framework, libSpina provides several preprocessor macros for encapsulating tasks such as exception handling and timeout-based error detection. Some checkpointing and message logging capabilities are available, but mostly for the initialization steps of FT-GENE. The use of the library causes a negligible overhead (circa 3%).

Since libSpina is just a tool designed to assist in programming, it does not provide any specific data recovery mechanism. Whenever a fault occurs, the lost data must be recovered by checkpoint/rollback strategies or an algorithmic/approximate recovery, which is not addressed by the library.

FT-GENE inherits the exception handling mechanism of libSpina and is fully written using non-blocking MPI calls thanks to libSpina's macros. Additionally, FT-GENE is able to withstand faults and provides two recovery methods: checkpoint/rollback and blank recovery (purely for testing purposes) which are discussed later.

3 Implementation

3.1 Parallel Implementation of the Combination Technique

The main parallelization strategy of our combination framework is the *manager-worker approach*. In this strategy, a dedicated process—the manager—distributes work packages to workers which then process their tasks. Within the combination technique these tasks consist of solving the respective PDE on a specific component grid. Since these grids will usually be still too large to be solved on one MPI rank, we assign the tasks to a group of workers which we will call a *process group*. To

www.dbooks.org

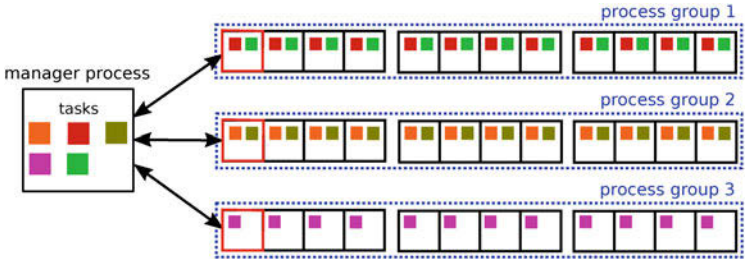


Fig. 3 The manager-worker implementation in our combination framework [13]. The manager assigns tasks to process groups which perform a domain decomposition to work on the tasks in parallel

avoid a communication bottleneck at the manager, only one dedicated rank in each process group—the master—communicates with the manager. This master process then broadcasts all information from the manager to the remaining processes in the process group. In Fig. 3 one can see an example of the task distribution to individual process groups.

The main concern of our framework is to achieve high parallel efficiency for potential exascale computing. Due to the independence of the component grids, we can solve all tasks independently. Additionally, all process groups compute each task in parallel by applying domain decomposition. This results in two levels of parallelism: parallelism between process groups and within process groups.

Unfortunately, in a chaotic time-dependent setting, we cannot completely decouple all tasks, as the individual solutions would eventually drift too far apart from the exact solution resp. from each other, which would deteriorate the result of the final combination. To avoid this effect, frequent recombination is applied (see Fig. 4). Here, we construct the sparse grid solution after a short simulation interval by gathering all component grids, and then redistribute the aggregated information to proceed with the computation. This process introduces a global synchronization point as well as global communication. To ensure high parallel efficiency, we

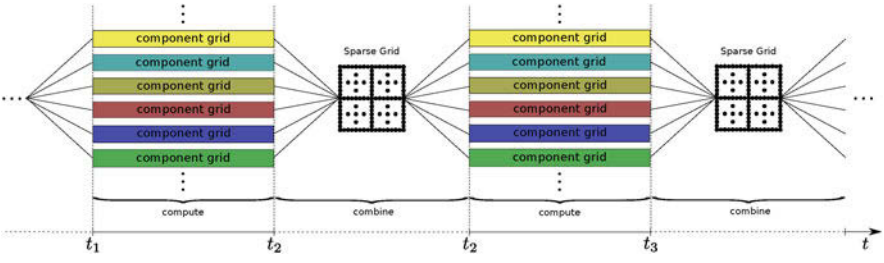


Fig. 4 Solving time-dependent PDEs with the combination technique [13]. Frequent recombination is applied in between computation phases to avoid that the individual solutions drift too far apart from each other

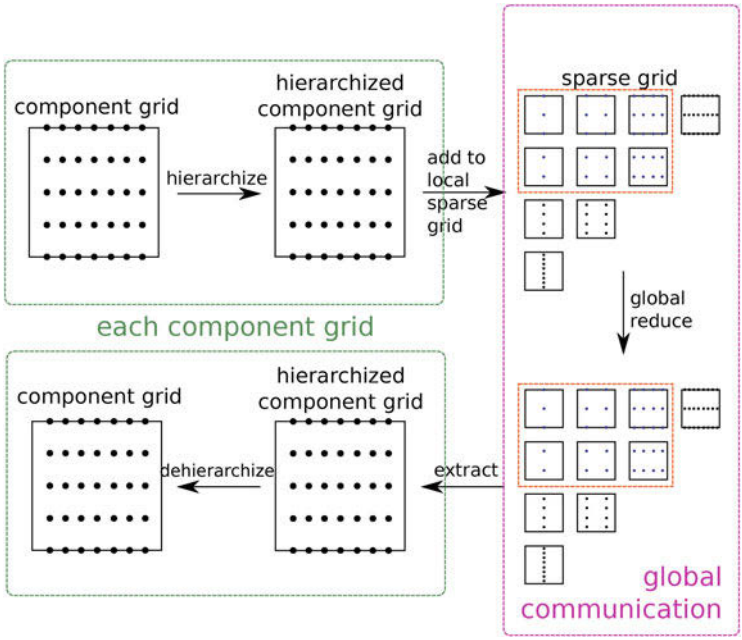


Fig. 5 The recombination step in detail [13]. After the hierarchization phase (top left), the sparse grid is globally reduced between process groups (right) and then dehierarchized within each process group (bottom left)

therefore require efficient load balancing (see Sect. 3.2) as well as an efficient recombination mechanism.

The recombination (Fig. 5) is split into three phases: hierarchization, global reduction of the sparse grid and dehierarchization. In the hierarchization step, each process group transforms the local solutions from its assigned tasks into the hierarchical basis. These hierarchized solutions are then added locally to the respective subspaces of the corresponding sparse grid. To this end, each process holds the part of the sparse grid that corresponds to their geometric subdomain in a buffer.

Thereafter, the global communication phase begins in which the partial sparse grid solutions from all process groups are added together with an *MPI_Allreduce*. In order to achieve low communication overhead, all process groups are constructed from the same number of ranks and apply the same domain decomposition on the sparse grid. A rank therefore directly communicates only with the ranks of other process groups that are assigned to the same subdomain. This procedure heavily reduces the number of communications and ensures a low communication overhead. Once the global communication has finished, the individual process groups extract the hierarchical information for their component grids from the sparse grid. The procedure is concluded by the dehierarchization step which transforms the data

www.dbooks.org

back to the nodal basis. It should be noted that only the second phase requires global communication while the hierarchization and dehierarchization only require communication within each process group.

3.2 Load Balancing

Load balancing for HPC simulations is routinely implemented via different standard techniques, such as domain decomposition or resource assignment. With the massively parallel distributed combination technique framework, there is an algorithmic way of performing load balancing. Due to the manager-worker scheme with process groups, cf. Fig. 3, the manager is free to assign multiple component grids to different process groups (provided there is enough free memory). In particular, this assignment can be chosen to balance the runtimes such that process groups do not have to wait for each other longer than necessary. An illustration is given in Fig. 6.

Of course, this assignment works best if accurate estimates of the individual grid runtimes can be obtained beforehand; then, we can even statically assign the grids. Reaching further, this can be improved using the information obtained in the first solver run: by collecting the runtimes of the (presumably) longest-running grids, the process groups can be "filled" with work. The initial dynamic "work-stealing" approach was presented by Heene [14]. The next step, dynamic reassignment according to load imbalances has not been implemented so far, since the reassignment of a grid to another process group could become very costly. In addition to the field grid data, extensive simulation data is required for GENE, such as the gyromatrix; it would have to be transferred or explicitly recomputed on the new process group. Now, using the estimates obtained by the model, a decreasingly ordered list can be created, and the grids can be assigned to the process group. This means that the relative ordering between tasks is more important than the absolute accuracy of the model.

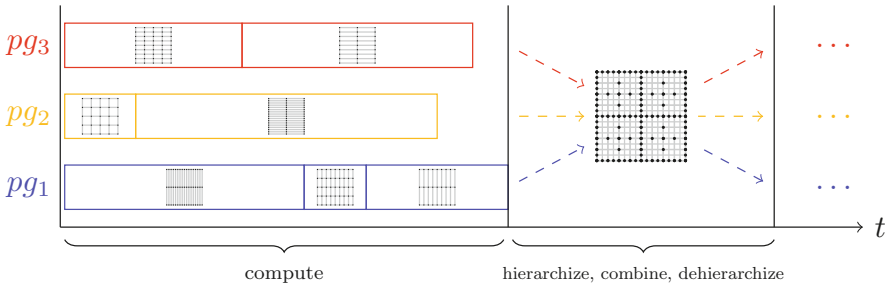


Fig. 6 Possible parallel computation scheme for different 2D grids, cf. [22]. In principle, the hierarchization step for a task could happen immediately after its completion, further improving load balancing. This is not included in our implementation to date, but this does not proof to be a significant bottleneck

The simplest approach would be to just take the number of grid points into account [9], assuming that all grids of the same level sum $|\boldsymbol{\ell}|_1$ take the same time to run. In larger scenarios, this will lead to variable results, as the assignment will be ambiguous. A more expert knowledge based model can be designed by adding a dependence on grid anisotropy [14]. In this model the runtime contribution of the number of grid points N is estimated with a function $r(N)$,

$$r(N) := mN^k + b \,, \tag{10}$$

where the coefficients m, k and b are fitted to runtime data of isotropic grids. Moreover, the influence of anisotropy is added through an additional term $h(\mathbf{s}_\ell)$, where $s_{\ell,i} = \frac{\ell_i}{|\boldsymbol{\ell}|_1}$. The total runtime estimate $t(N, \mathbf{s}_\ell)$ then reads

$$t(N, \mathbf{s}_\ell) = r(N) \cdot h(\mathbf{s}_\ell) \qquad h(\mathbf{s}_\ell) := c + \sum_{i=1}^{d-1} c_i s_{\ell,i} + \sum_{i=1}^{d-1} \sum_{j \le i} c_{ij} s_{\ell,i} s_{\ell,j} + \dots \,. \tag{11}$$

Again, the coefficients c in the polynomial ansatz for $h(\mathbf{s}_\ell)$ are fitted to runtime data, this time including anisotropic grids.

For nonlinear, global simulations, the load modeling was recently extended to fully data-driven techniques, such as support vector regression and neural networks [22].

3.3 Fault Tolerant Combination Technique

Fault tolerance is becoming more and more important in exascale frameworks as the increasing process number will most certainly also increase the probability for some components to fail. In our case we assume that such an error will most likely occur during the computation phase, which takes the majority of the overall runtime. The computation step will therefore not complete for all affected process groups. In the next section we briefly summarize the progress made in [15, 16, 19, 20].

Such faulty groups are detected by the manager at the beginning of the global communication phase with our fault simulation layer. This simulation layer imitates the behavior of ULFM and returns an error signal if a process has failed that should interact in a certain communication. It is also possible to simulate the process failure of specific ranks at specific simulation points. Once the affected groups are known, the manager process calculates a fault-free combination scheme by solving the GCP (cf. Sect. 2.3.1). We use GLPK [5] to solve this optimization problem which returns the new combination scheme.

Unfortunately we cannot proceed directly with the communication at this point since failed ranks are contained in the MPI communicators. The naive implementation would just remove all failed process groups and continue computation on

www.dbooks.org

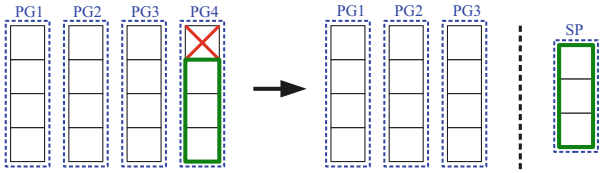


Fig. 7 If a process failure occurs in one of the process groups, the remaining ranks are declared as spare processes [19]

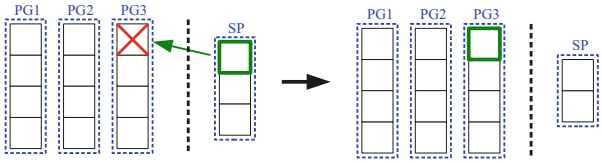


Fig. 8 In case all failed ranks can be replaced by spare processes, the process group is restored [19]

the remaining groups. This procedure, however, might waste valuable resources if only few of the ranks in a process group have failed. We therefore save these non-faulty ranks and declare them as spare ranks (Fig. 7). For future faults we can now substitute failed ranks by spare ranks to restore a process group (Fig. 8). This enables us to simulate high failure rates without quickly losing all process groups.

Once the MPI environment is restored, our implementation proceeds with the recombination step by applying the global communication to the fault-free combination scheme. For future computations we restore the original combination scheme by redistributing failed tasks to the remaining process groups. These tasks are then initialized by extraction from the sparse grid that results from the temporary fault-free combination scheme.

4 Numerical Results

4.1 Convergence

A very important task in order to confirm the applicability of the sparse grid combination technique to GENE is to show that we can produce meaningful results while reducing the computational cost. This has to be verified in different ways and for different quantities, depending on the setting of the simulation. Therefore, this section is split into two parts.

The first one is concerned with convergence results for local linear GENE runs, where the accuracy of computations of the growth rate $\lambda_{\max}$ and its corresponding eigenvector $\mathbf{g}^{(\lambda_{\max})}$ is studied. The second part presents results for nonlinear GENE

simulations, in particular the most recent investigations of nonlocal runs. Here, one is interested in time averages of certain quantities of interest during the quasi-stationary phase, mainly the mean heat flux Q_{es}.

4.1.1 Linear Runs

Linear simulations only use the linear part of the gyrokinetic equations,

$$\frac{\partial \mathbf{g}}{\partial t} = \mathcal{L}\,\mathbf{g}\,, \tag{12}$$

according to Eq. (9) (the bold face $\mathbf{g}$ signifies the discretized version). The dynamics of this equation can be understood by considering an eigenvalue decomposition (assuming every eigenvalue has multiplicity one)

$$\mathbf{g}(t) = \sum_{\lambda \in \sigma(\mathcal{L})} \alpha^{(\lambda)}(t)\,\mathbf{g}^{(\lambda)} \qquad , \qquad \mathcal{L}\mathbf{g}^{(\lambda)} = \lambda\,\mathbf{g}^{(\lambda)}\,. \tag{13}$$

Note that $\mathbf{g}$ is complex valued because a Fourier basis is always used in the y-direction and, hence, eigenvalues will be complex in general.

Plugging this decomposition into Eq. (12) yields the solution

$$\mathbf{g}(t) = \sum_{\lambda \in \sigma(\mathcal{L})} \alpha^{(\lambda)}(0)\,e^{\lambda t}\,\mathbf{g}^{(\lambda)}\,. \tag{14}$$

The exponential growth (or decay) will, for large times, be dominated by the eigenvector corresponding to the eigenvalue $\lambda_{\max}$ which has the largest real part. This growth rate plus eigenmode are the quantities one is interested in. To this end, GENE can be run either as a direct eigenvalue solver or as an initial value solver. For the latter case, one initializes a random initial state and simulates long enough, until the shape of $\mathbf{g}(t)$ stays constant and its ratio at subsequent times settles on $e^{\lambda_{\max}\Delta t}$.

First results for the linear case were obtained during the first funding period of this project. The combination technique was successfully applied to both the eigenvalue solver and initial value runs. In each case, combining the eigenvalues as well as the eigenvectors produced by the component solutions yielded satisfactory results. We refer to [18] and omit the details here.

Instead we want to focus on results obtained for local linear initial value runs with the massively parallel framework described in Sect. 3.1, using the possibility of frequent recombination after short time intervals. The first studies with this framework were conducted during the first half of the second funding period and are recorded in [13]. They examined the convergence of the eigenvector in a GENE test case simulating ion temperature gradient (ITG) driven instabilities.

The following combination schemes were set up for this experiment, always using the z-, $v_\parallel$- and μ-directions for combination (the k_x- and k_y-direction were

www.dbooks.org

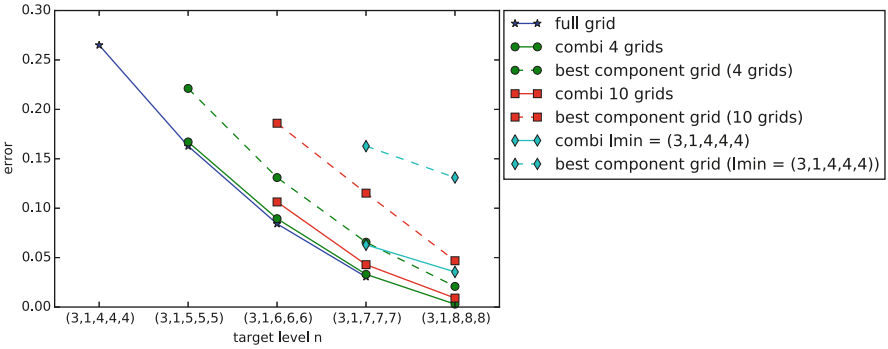


Fig. 9 Error of $\mathbf{g}^{(\lambda_{\max})}$ compared to the reference solution with $\boldsymbol{\ell} = (3, 1, 8, 8, 8)$. The combined solutions were obtained by combining after each time step with a total of 6000 time steps

fixed at level 3 and 1, respectively, as not much resolution is needed in these dimensions in local linear settings): The terms "combi 4 grids" and "combi 10 grids" describe schemes where $\boldsymbol{\ell}_{\min}$ and $\boldsymbol{\ell}_{\max}$ are $(0, 0, 1, 1, 1)$ resp. $(0, 0, 2, 2, 2)$ apart. This results in the mentioned number of component solutions. Finally, "combi lmin" means that $\boldsymbol{\ell}_{\min} = (3, 1, 4, 4, 4)$ is kept fix regardless of maximal level. In order to calculate the error of the obtained eigenvectors, a reference solution was computed on a high resolution full grid with $\boldsymbol{\ell} = (3, 1, 8, 8, 8)$. All other solutions were interpolated to this reference grid and normalized to unity (because only the shape matters), then the L_2-norm of the difference in absolute value was taken as the error measure.

Figure 9 shows results for all combination schemes as well as single full grid solutions for comparison. Here, a recombination was applied after each GENE time step, which turned out to yield the best overall results. One can see that the error decreases with higher target levels and that the combined solutions compare well to the full grid convergence. The higher the number of component grids, the worse the error as interpreted at the target level, but keep in mind that the number of degrees of freedom is more and more reduced. All combination schemes also beat the best error achieved with one of their component grids, which is a crucial test, since otherwise one could be content with that one component solution.

We also have to show that the computational cost is reduced. To this end, different combination intervals (in number of time steps) have been investigated as shown in Fig. 10. It turns out that the overhead for recombination after each time step was too large. There are two main reasons. Firstly, this test case was comparatively small so that the relative overhead is more significant than for much larger problem sizes. Secondly, the restart behavior of GENE (as discussed in Sect. 2.2) is still not optimal so that the pure GENE runtime is much larger for frequent recombination. However, the figure also shows that there is a middle ground. For example, the run time of "combi lmin" with $\mathbf{n} = (3, 1, 8, 8, 8)$ at a combination interval of 10 is already

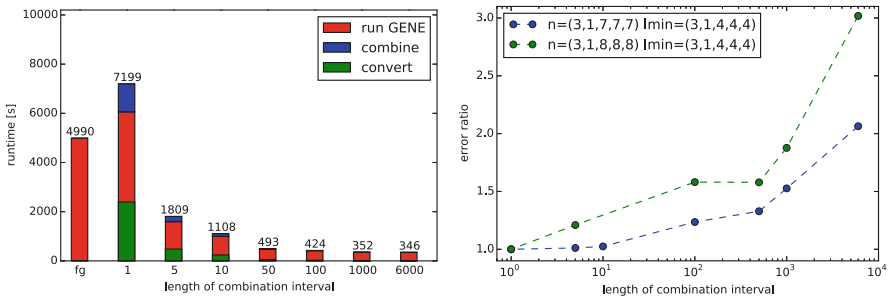


Fig. 10 **Left**: Computation times for "combi lmin" with $\mathbf{n} = (3, 1, 8, 8, 8)$ at different combination intervals and for the reference full grid solution. Four process groups with 16 processes each were used. **Right**: Relative increase of the error for $\mathbf{g}^{(\lambda_{\max})}$ for different combination intervals

lower by a factor 5 than that of the reference solution. At the same time, the error is only increased by about 25% compared to the optimum.

In conclusion, the applicability of the combination technique with the massively parallel framework has been demonstrated for a small test case. The benefit is expected to be even bigger for larger problem sizes.

4.1.2 Nonlinear Runs

In nonlinear simulations the initial exponential growth phase is halted after some time by the increasingly dominant influence of the nonlinear part of the PDE. Henceforth, the distribution function is subjected to chaotic dynamics. This means that slight disturbances will completely alter the trajectory in the long run (butterfly effect) and, thus, the distribution function itself is not a suitable observable anymore. Still, distinctive patterns in a typical trajectory can be observed that result in a quasi-stationary state. Common observables (often averages or moments of the distribution function) will statistically fluctuate around relatively robust mean values that only depend on the problem parameters. Hence, the goal of the combination technique should be to accurately reproduce said quantities of interest with a similar statistical uncertainty. Throughout this chapter, an ITG test case with adiabatic electrons is studied. We switched to nonlocal simulations to increase the problem size in line with the conclusion of the previous section.

In order to familiarize ourselves with the behavior of important quantities of interest, we initially tested directly combining them instead of g. That is, we choose a combination scheme and, once all component runs are finished, we calculate the time averages and linearly combine them with the appropriate combination coefficients. The method we used to determine when the quasi-stationary phase begins and to estimate the statistical uncertainty follows [25].

We focused on the mean heat flux Q_{es} because it is one of the most statistically robust quantities. Results for various combination schemes in different dimensions

www.dbooks.org

Table 1 Results for direct combination of Q_{es}

$\boldsymbol{\ell}_{\max}, \boldsymbol{\ell}_{\min}$	Reference	Direct combination	Best component
(**10**,5,**6**,5,4), (**7**,5,**3**,5,4)	21.7 ± 0.4	23.1 ± 0.9 (6% 0.66)	23.7 ± 0.4 (9% 0.13)
(**9**,5,**5**,**6**,**5**), (**6**,5,**3**,**4**,**3**)	20.2 ± 1.0	28.2 ± 8.1 (40% 0.33)	29.9 ± 0.5 (48% 0.04)
(**11**,5,**5**,**7**,**4**), (**7**,5,**4**,**4**,**3**)	21.8 ± 0.5	21.6 ± 1.4 (1% 0.37)	22.8 ± 0.4 (5% 0.04)

The two values in parentheses denote the relative error and the fraction of work load (in core-hours) in relation to the reference solution

are summarized in Table 1. The reference values in each case were computed with a run at target resolution. Unfortunately, adding and subtracting statistically independent quantities causes the variance of the resulting quantity to be the sum of variances of the summands scaled by the square of their combination coefficient. Since we take the standard deviation as a measure of uncertainty, we get $\sigma^{(c)} = (\sum_i c_i^2 \sigma_i^2)^{1/2}$. This is why the uncertainty for the second scheme became very high. Furthermore, the combined values are often not significantly better than the best component solution.

The problem is that finding an efficient level set is a balancing act. On the one hand, the minimal level is restricted by GENE simulations becoming erroneous for too low resolutions when physically relevant scales are no longer resolved. On the other hand, the maximal level is bounded by running into the statistical uncertainty so that an increase in accuracy is wasted. This could be addressed by including the simulation length as another dimension in the combination technique but decreasing the uncertainty is always expensive since it scales only with the inverse square root of the simulation length. On top of all this, there is a large discrepancy in the influence of different dimensions, in particular, the error is mostly dominated by the resolution in x-direction. The third scheme was chosen in a fashion to address these issues and it shows the best agreement with the reference. We currently also work on optimizing index sets with a *dimensionally adaptive* algorithm [4] to further mitigate said issues.

Recently, we obtained first results with the massively parallel framework applied to the nonlinear test case. In this context we turned back to frequently recombining g even in the nonlinear regime. Much care has to be taken since the trajectories on different component grids have the tendency to drift apart fairly quickly. Thus, the combination interval should be chosen sufficiently small. However, in contrast to the linear case, we found that recombining after each or just a few time steps leads to the simulation becoming unstable and the distribution function growing uncontrollably. We had to increase the combination interval until stable trajectories were achieved. Also, depending on which and how many dimensions were included in the combination scheme, simulations could become unstable. The causes of these effects are not yet understood and currently under investigation. We suspect that the perturbation introduced by distributing the combined checkpoint to the component grids is large enough so that each trajectory needs a certain relaxation time to reach the quasi-stationary state again.

Table 2 Results for frequent recombination of g

$\ell_{\max}$, $\ell_{\min}$	Reference	Recombination of g	Direct combination
(**10**,5,**5**,4,3), (**8**,5,**3**,4,3)	27.3 ± 0.6	28.4 ± 0.5 (4%)	27.4 ± 1.7 (0.3%)
(**11**,5,**5**,4,3), (**9**,5,**3**,4,3)	28.1 ± 1.1	24.9 ± 0.5 (11%)	27.9 ± 2.2 (0.7%)
(**10**,5,3,**6**,3), (**8**,5,3,**4**,3)	27.3 ± 1.6	27.9 ± 0.9 (2%)	28.4 ± 1.4 (4.0%)

The values shown are the mean heat flux Q_{es} with relative error to the reference in parentheses. The alleged outlier is actually closer to the high resolution reference from Table 1

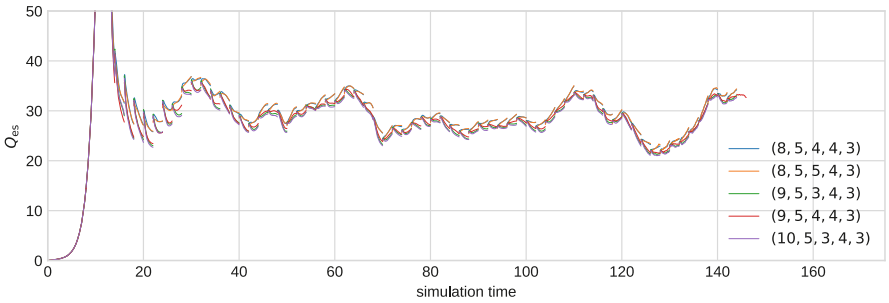


Fig. 11 Time traces of the mean heat flux Q_{es} for the five component grids of the first scheme in Table 2. One sees the effect of the recombination of g after every two units of time

Nevertheless, results for three 2D combination schemes with a recombination interval of two units of simulation time are presented in Table 2. The trajectories of Q_{es} for the different component grids are shown in Fig. 11. They look promising since one can observe that after a combination step the values "snap" to a similar value. The remaining discrepancy is solely caused by the influence of different resolutions during the calculation of Q_{es}. There is no functionality yet to directly compute this quantity on the underlying sparse grid because its implementation would have been too time-consuming. Instead, the values at the combination times were gathered and again linearly combined according to the combination scheme. With this procedure we achieved results that are in accordance to their reference solution. Still, the same considerations as for the one-time combination apply here and we look forward to expanding the tests to more optimized index sets in the future.

4.2 Scaling Analysis

Scalability is a crucial aspect for an exascale-ready framework. For our framework this boils down to two aspects: achieving a good load balance to avoid idling process groups and keeping the overhead of recombination low as the main computation happens in the black box solver. We expect that the black box solver itself already

www.dbooks.org

provides good scalability up to a certain number of cores. By parallelizing over many tasks we can then boost this scalability to reach process counts beyond the capabilities of the solver. This is based on the fact that the number of cores can be scaled in two ways: increasing the number of cores in the process groups—scaling the solver—or increasing the number of process groups while keeping their core numbers equal.

We will first look at the scalability of the recombination steps [13] for a linear test case with $\boldsymbol{\ell}_{\max} = (14, 6, 6, 8, 7)$ and $\boldsymbol{\ell}_{\min} = (9, 4, 4, 6, 4)$ on Hazel Hen (HLRS). In Fig. 12 we can see that the hierarchization scales almost perfectly with the number of cores as it does not involve computation between process groups. Moreover, it mainly involves local computation with little communication in the group. Dehierarchization performs almost identically as it is the inverse function of the hierarchization. The local reduction step only adds the hierarchical coefficients of the local component grids to the sparse grid. As we enforce the same domain decomposition of the sparse grids and the component grids, this introduces no communication. Consequently, this operation scales perfectly. The global reduction of the sparse grid, however, involves the communication of the decomposed sparse grid parts over the process groups. Therefore, it shows no scaling behavior but also a comparably low runtime.

If we now sum up all components of our recombination step (bottom graph in Fig. 12), we can see that we can scale up to the whole size of the Hazel Hen supercomputer if the number of process groups is adjusted well. As a consequence, the framework introduces only a low overhead compared to the computation time of GENE itself.

This observation is confirmed by the measurements taken for nonlinear runs on Hazel Hen, shown in Fig. 13. This scenario consisted of 16 grids at relatively high resolutions. The optimal linear scaling is reached for moderate numbers of processes, yet the total scaling capabilities are limited by the solver's scaling capabilities. We see again that the sparse grid combination technique can play out its advantages only if the scenario is chosen to contain many grids, i.e., to have significant spans between $\boldsymbol{\ell}_{\min}$ and $\boldsymbol{\ell}_{\max}$. Even then, for our use cases, the overhead imposed by the combination routines is negligible compared to the total runtime.

4.2.1 Load Balancing

We will see here that, using the algorithmic opportunity for load balancing by grid assignment based on a good load model, cf. Sect. 3.2, we can achieve good parallel efficiency for large simulations—applied to both linear and nonlinear GENE runs.

The linear and anisotropy-based models were tested on 32 cores [14]. For this linear experiment, the scenario was constructed from $\boldsymbol{\ell}_{\min} = (3, 1, 3, 3, 3)$ to $\boldsymbol{\ell}_{\max} = (11, 1, 11, 11, 11)$, containing 425 grids. Figure 14 shows that the initial dynamic approach ("work stealing") is preferable to the static assignment, because high parallel efficiencies can be sustained as the number of process groups grows.

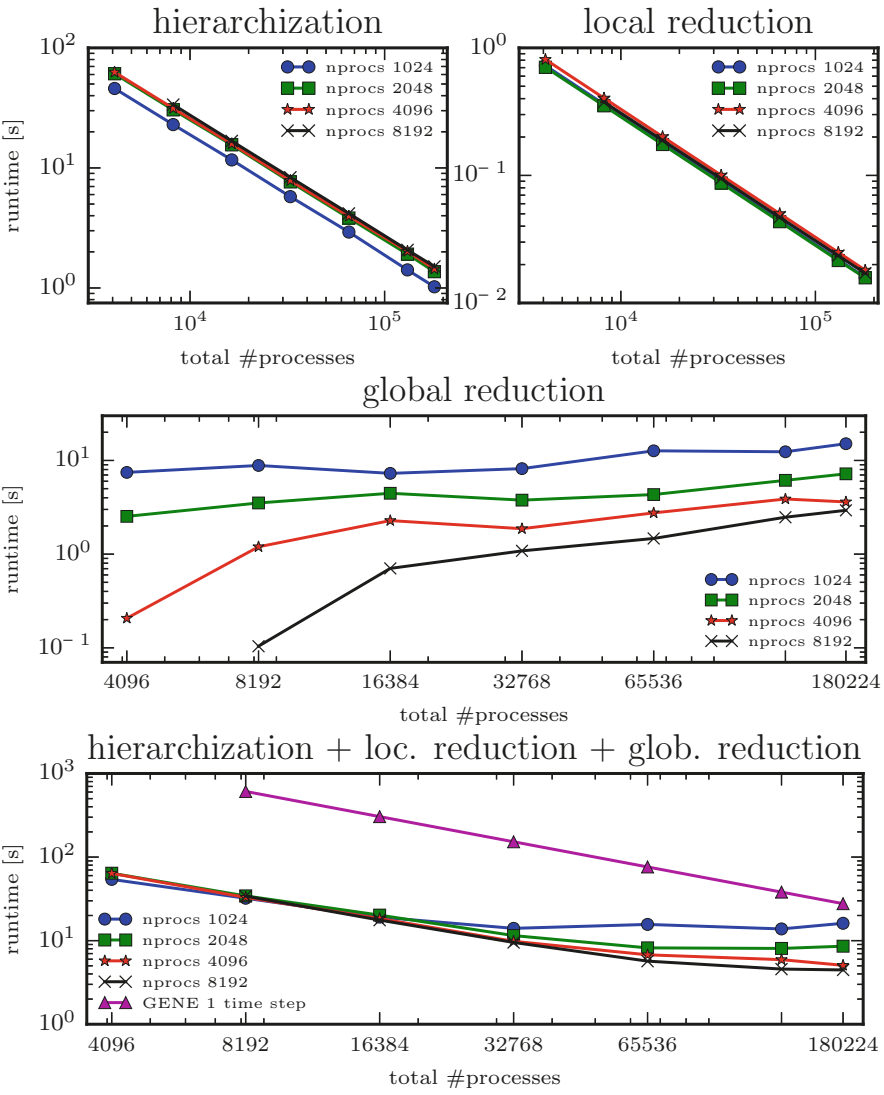


Fig. 12 Timings for the individual steps of the distributed recombination step for different sizes of the process groups. The bottom plot shows the total time of hierarchization, local reduction and global reduction. We also included a rough estimate of the computation time for one time step of GENE with process groups of size 4096. Graphs from [13]

We can also conclude that taking the anisotropy into account leads to, on average, substantial improvements with respect to the parallel efficiencies.

In order to cope with the larger-scale nonlinear GENE simulations, a similar methodology was used to evaluate data-driven techniques. The methods compared

www.dbooks.org

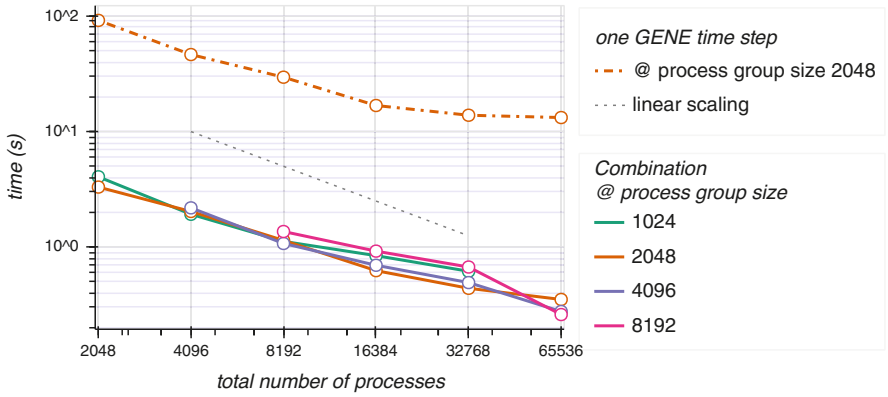


Fig. 13 Timings for a combination scheme computed with the nonlinear GENE functionality: ITG with adiabatic electrons, for $\boldsymbol{\ell}_{\min} = (7, 1, 5, 6, 4)$ to $\boldsymbol{\ell}_{\max} = (10, 1, 7, 8, 6)$, averaged over ten combination time steps on Hazel Hen

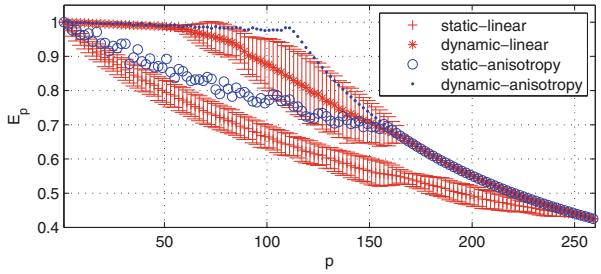


Fig. 14 Parallel efficiency for the linear ITG scenario: linear vs. anisotropy-based, static vs. linear

were, again, the anisotropy-based model, nearest neighbor estimates, support vector regression (SVR) and neural networks [22]. These models were trained on 2048 randomly sampled tasks. As a reference, the best attainable balance that could be produced by the dynamic filling heuristic (cf. Sect. 3.2)—through an estimate that is the same as the true runtime—is included in the comparison. This time, the scenario contained grids from $\boldsymbol{\ell}_{\min} = (7, 4, 3, 4, 3)$ to $\boldsymbol{\ell}_{\max} = (12, 8, 6, 8, 6)$ to account for the anisotropy in the requirements for different solutions, comprising 237 grids. Now, the process group size was not fixed to 32 any more, but could reach up to 2^{15} or 32,768 processes.

It is to be noted that Fig. 15 shows regions of high parallel efficiency compared to Fig. 14. Still, the differences between the different modeling methods are significant: while the nearest neighbor and SVR models lead to a relatively quick degradation of parallel efficiency, the expert knowledge-based anisotropy model performs a lot better, and the neural network can even achieve near-optimal scaling (the improvements over the optimal estimate are of course lucky guesses). This

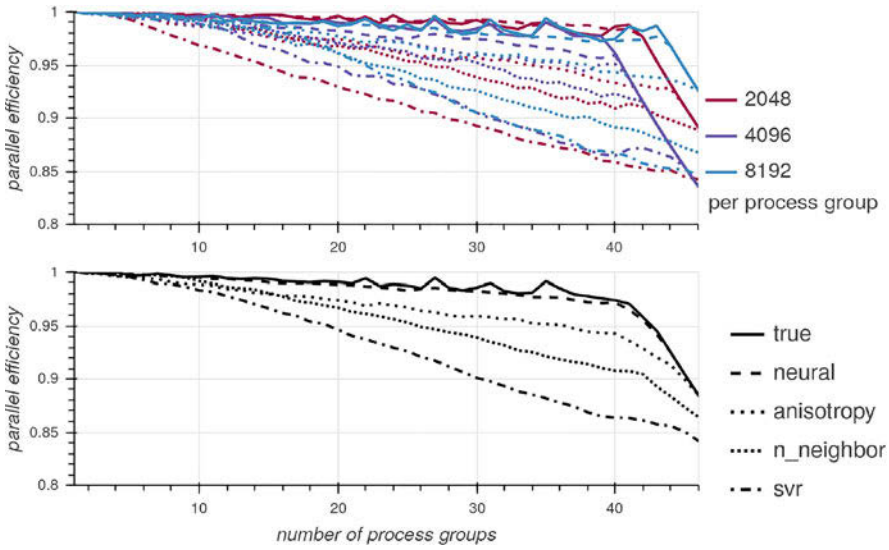


Fig. 15 Parallel efficiencies for the nonlinear, larger ITG scenario: by method and process group size (upper), and averaged over the different process group sizes for clarity (lower). The legend of the lower plot applies to the upper one in the same way

advantage is enabled by having enough data samples available. Since GENE is routinely run on various HPC machines, strong load models could be generated even across machines from the runtime data through neural networks.

4.3 Fault Tolerance

4.3.1 Fault Tolerant Combination Technique

This chapter briefly summarizes the most recent results with hard faults within the FTCT from [19]. For further results on hard and soft faults see [15, 16, 20].

To simulate statistical effects on the accuracy with faults in the FTCT, we conduct a random sampling for generating process failures during the simulation. For this sampling we choose the commonly used Weibull distribution [23]:

$$f(t;\lambda,k) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-(t/\lambda)^k} \tag{15}$$

where k and λ are the shape and scale parameter. λ is used to directly control the failure rate where smaller λ values cause larger error rates. This distribution is used to draw a failure time after which an MPI rank will fail. E.g., a failure time of 10s means that after a wall clock time of 10s a process will fail.

www.dbooks.org

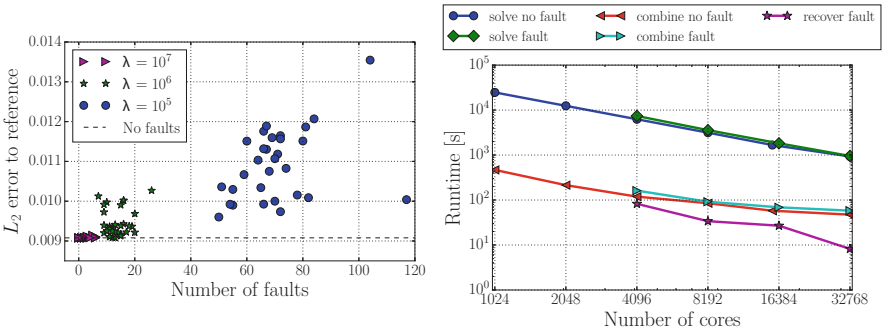


Fig. 16 **Left**: Statistical results for the L_2 error of the FTCT with different numbers of faults [19]. The errors stay relatively small, even in the presence of around 100 faults the error increases by less than a factor of two. **Right**: Average runtimes for our scaling run with the FTCT [19]. One can see that the overhead of the fault recovery is negligible compared to the runtime of the solver

We show results for one linear test case with a 3D combination in z,μ and $v_\parallel$ with $\boldsymbol{\ell}_{\max} = (8, 8, 8)$ and $\boldsymbol{\ell}_{\min} = (5, 5, 5)$. The remaining dimensions x an y are fixed at 9 and 1 grid points, respectively. Statistical results for 100 individual runs with 512 MPI ranks are shown in Fig. 16 (left). It can be seen that for lower failure rates the error is almost identical to the base line error without any process failure (dotted line). In case of larger failure rates the accuracy decreases but the overall error increase is still tolerable compared to the number of failing ranks. On average, the error increases by 0.09%, 4% and 20%, respectively, for lambda values of 10^7, 10^6 and 10^5.

Apart from error analysis, we are interested in the scaling behavior of the FTCT in the presence of faults. For this analysis we again choose a 3D combination in z,μ and $v_\parallel$ with $\boldsymbol{\ell}_{\max} = (14, 14, 14)$ and $\boldsymbol{\ell}_{\min} = (4, 4, 4)$. In this case we simulate only 300 time steps and apply 3 recombinations, i.e. every 100 steps, and inject a single fault in one of the process groups. The process group size stays fixed at 1024 cores and only the number of process groups is varied in the scaling experiment (Fig. 16, right).

In general we observed a very good scaling behavior even in the presence of faults. Of course, the runtimes are slightly increased due to the fault of one process group, which reduces the overall process count after the fault occurs. However, the scaling properties seem to be unaffected by the fault recovery. This can also be seen in the low overhead of the recovery step which is about two orders of magnitude lower than the time for solving the PDE. Also the time for the recombination scales well even though it involves global communication.

All in all, our results with linear GENE simulations have shown that we can achieve fault tolerance with small computational overhead while keeping a similar accuracy. For further details about the experiments and results we refer to [19].

4.3.2 libSpina

We answer here two questions concerning FT-GENE: how its performance compares with the performance of standard GENE and what kind of faults the framework is able to recognize and tackle. The clusters Hazel Hen, Cobra and Draco (MPCDF) were used in the tests which follow.

Robustness These tests used realistic parameters (based on the benchmark ITGTEM) where a random subset of the MPI ranks is selected to fail. Several combinations of faults were tested (full node failure, partial node failure, multiple failures in different time steps, etc.) as well as different causes for failure (e.g. "stop" intrinsic, or externally calling the "pkill" command).

The library libSpina was able to detect the faults, notify FT-GENE, restore the MPI environment and allow FT-GENE to read the last checkpoint and rollback the computation. Not all fault types could be handled by libSpina, most notably faults caused by interrupting the interconnect hardware. Otherwise, the results were consistent and reproducible.

Performance and Overhead The aim of these tests was to determine the performance impact of libSpina on an application, by running both FT-GENE and GENE "back-to-back" on the same hardware with the same parameter files.

In the following, the grid dimensions are displayed as $(x, y, z, v_{||}, \mu)$. The same notation is used for the MPI partitioning of the domain (that is, n_proc_x, n_proc_y, and so on). The runtime is shown in seconds and disregards the initialization of GENE and libSpina. The reason is that the initialization of GENE involves several performance optimization tests (often lasting several minutes) which would mask the true difference in runtime.

The first part of the results in Table 3 has been repeated many times for many different configurations: nonlocal and local, linear and nonlinear, between 1 and 64

Table 3 Cobra performance comparison

Nodes	μ	n_proc_μ	GENE	FT-GENE	Loss
2	2	1	379.2 s	386.1 s	1.83%
4	4	2	398.9 s	400.7 s	0.46%
8	8	4	419.6 s	422.4 s	0.67%
16	16	8	414.5 s	416.2 s	0.40%
32	32	16	424.4 s	432.4 s	1.89%
64	64	32	438.0 s	451.4 s	3.06%
150	5	5	568.4 s	565.3 s	−0.55%
300	10	10	590.4 s	585.0 s	−0.91%
600	20	20	714.6 s	700.4 s	−1.99%

Topmost: (512,64,40,*,48) points, (8,1,5,*,1) parallelization, nonlocal, kinetic electrons (2 species). Lowermost: (512,512,20,*,60) grid size, (1,2,20,*,15) parallelization, local, kinetic electrons. Both are nonlinear runs with 100 time steps and 40 ranks per node

www.dbooks.org

Table 4 Hazel Hen performance comparison

Nodes	x	n_proc_x	GENE	FT-GENE	Loss
1024	576	24	517 s	446 s	−13.7%
1024	576	24	469 s	477 s	1.7%
2048	1152	48	1156 s	805 s	−30.4%
2048	1152	48	836 s	914 s	9.3%

Grid size: (*,128,32,64,16) points, parallelization: (*,1,8,8,16), nonlocal, adiabatic electrons (1 species), nonlinear run, 1000 time steps, 24 ranks per node

nodes, in Cobra and Draco with several different types of parallelization. On all occasions the result was consistent and the overhead of libSpina lay between −3% and +3% which is within statistical fluctuations in this case. For the sake of brevity, only the most recent of these results are shown in Table 3.

There was no deterioration of the performance of FT-GENE for large numbers of nodes, as it can be seen in the lowermost part of Table 3.

Similar experiments were performed on Hazel Hen but provide largely inconclusive results. Small tests (with four nodes) were analogous to the results shown in Table 3, but tests involving a large number of resources were inconclusive (see Table 4). The overhead fluctuated between −30% and +15%. One of the possible explanations is that the job was distributed amongst several islands and MPI communications had to be passed through switches. Since switches are shared amongst all running jobs, the communication patterns of GENE and FT-GENE were disturbed by external factors, causing statistically invalid measurements of performance.

5 Conclusion

During the EXAHD project a massively parallel software framework for the sparse grid combination technique was developed and successfully applied to the gyrokinetic solver GENE, to linear as well as nonlinear, nonlocal settings. While its functionality has been demonstrated for medium sized test cases we expect an even greater benefit when applying it to larger simulations in future experiments.

Our implementation shows excellent scaling behavior up to 180k cores, and allows to scale the application code beyond its specific capabilities, thus making it ready for exascale computing. At the same time the overhead incurred to the total runtime is negligible.

The load balancing that can be realized with our approach allows to maintain parallel efficiencies close to the optimum. This is achieved by incorporating effects of anisotropy in our cost model and applying dynamic task scheduling. By modeling the solver loads with neural networks, this scheduling can be further enhanced. As an outlook, the additional level of parallelism offered by the combination technique may allow for scaling beyond a single system: the computation may be

decoupled across compute centers. This would require more advanced distributed communication, as the combined solutions would have to be updated on both systems. First tests have been performed for this setup, but the overall performance and gains of such an approach still need to be evaluated.

We have also shown that the fault tolerant combination technique can be used to construct a resilient framework, which can tolerate hard and even soft faults. This algorithm-based fault tolerance introduces only minor effects on the overall accuracy and runtime while preserving the scaling properties of the framework. Furthermore, FT-GENE demonstrated a reasonable ability to tolerate faults, even under the restrictions of relying purely on standard MPI, causing no performance overhead.

Finally, by implementing specialized interfaces, the software framework can readily be extended to other codes dealing with high-dimensional functions and we look forward to seeing it used in many more applications.

References

1. Bland, W., Bouteiller, A., Herault, T., Bosilca, G., Dongarra, J.: Post-failure recovery of MPI communication capability: design and rationale. Int. J. High Perform. Comput. Appl. **27**(3), 244–254 (2013)
2. Bungartz, H.J., Griebel, M.: Sparse grids. Acta Numer. **13**, 147–269 (2004)
3. Bungartz, H.J., Griebel, M., Röschke, D., Zenger, C.: Pointwise convergence of the combination technique for the Laplace equation. East-West J. Numer. Math. **2**, 21–45 (1994)
4. Gerstner, T., Griebel, M.: Dimension–adaptive tensor–product quadrature. Computing **71**(1), 65–87 (2003)
5. GLPK (GNU linear programming kit). https://www.gnu.org/software/glpk/
6. Görler, T.: Multiscale effects in plasma microturbulence. Ph.D. thesis, Universität Ulm (2009)
7. Griebel, M., Hamaekers, J.: Sparse grids for the Schrödinger equation. Math. Model. Numer. Anal. **41**(2), 215–247 (2007)
8. Griebel, M., Harbrecht, H.: On the convergence of the combination technique. In: Sparse Grids and Applications. Lecture Notes in Computational Science and Engineering, vol. 97, pp. 55–74. Springer, Cham (2014)
9. Griebel, M., Huber, W., Rüde, U., Störtkuhl, T.: The combination technique for parallel sparse-grid-preconditioning or -solution of PDEs on workstation networks. In: Bougé, L., Cosnard, M., Robert, Y., Trystram, D. (eds.) Parallel Processing: CONPAR 92 VAPP V, LNCS, vol. 634. Springer, Berlin (1992)
10. Griebel, M., Schneider, M., Zenger, C.: A combination technique for the solution of sparse grid problems. In: de Groen, P., Beauwens, R. (eds.) Iterative Methods in Linear Algebra, pp. 263–281. IMACS, Elsevier, North Holland (1992)
11. Harding, B.: Fault tolerant computation of hyperbolic partial differential equations with the sparse grid combination technique. Ph.D. thesis, Australian National University (2016)
12. Harding, B., et al.: Fault tolerant computation with the sparse grid combination technique. SIAM J. Sci. Comput. **37**(3), C331–C353 (2015)
13. Heene, M.: A massively parallel combination technique for the solution of high-dimensional PDEs. Ph.D. thesis, Institut für Parallele und Verteilte Systeme der Universität Stuttgart (2018)
14. Heene, M., Kowitz, C., Pflüger, D.: Load balancing for massively parallel computations with the sparse grid combination technique. In: Parallel Computing: Accelerating Computational Science and Engineering (CSE). Advances in Parallel Computing, vol. 25, pp. 574–583 (2014)

www.dbooks.org

15. Heene, M., Parra Hinojosa, A., Bungartz, H.J., Pflüger, D.: A massively-parallel, fault-tolerant solver for high-dimensional PDEs. In: European Conference on Parallel Processing (2016)
16. Heene, M., Parra Hinojosa, A., Obersteiner, M., Bungartz, H.J., Pflüger, D.: Exahd: An exa-scalable two-level sparse grid approach for higher-dimensional problems in plasma physics and beyond. In: Nagel, W., Kröner, D., Resch, M. (eds.) High Performance Computing in Science and Engineering ' 17. Springer, Cham (2018)
17. Jenko, F., Dorland, W., Kotschenreuther, M., Rogers, B.: Electron temperature gradient driven turbulence. Phys. Plasmas **7**(5), 1904–1910 (2000)
18. Kowitz, C.: Applying the sparse grid combination technique in linear gyrokinetics. Dissertation, Technische Universität München, München (2016)
19. Obersteiner, M., Parra Hinojosa, A., Heene, M., Bungartz, H.J., Pflüger, D.: A highly scalable, algorithm-based fault-tolerant solver for gyrokinetic plasma simulations. In: Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (2017)
20. Parra Hinojosa, A., Harding, B., Hegland, M., Bungartz, H.J.: Handling silent data corruption with the sparse grid combination technique. In: Software for Exascale Computing-SPPEXA 2013–2015, pp. 187–208. Springer, Cham (2016)
21. Peherstorfer, B., Kowitz, C., Pflüger, D., Bungartz, H.J.: Selected recent applications of sparse grids. Numer. Math. Theory Methods Appl. **8**(1), 47–77 (2015)
22. Pollinger, T., Pflüger, D.: Learning-based load balancing for massively parallel simulations of hot fusion plasmas. Parallel Computing: Technology Trends, Proceedings of PARCO 2019. Advances in Parallel Computing, vol. 36, pp. 137–146. IOS Press (2019)
23. Schroeder, B., Gibson, G.: A large-scale study of failures in high-performance computing systems. IEEE Trans. Dependable Secure Comput. **7**(4), 337–350 (2010)
24. Smolyak, S.: Quadrature and interpolation formulas for tensor products of certain class of functions. Sov. Math. Dokl. **4**, 240–243 (1963). Russisches Original: Doklady Akademii Nauk SSSR, 148 (5): 1042–1053
25. Told, D., Cookmeyer, J., Muller, F., Astfalk, P., Jenko, F.: Comparative study of gyrokinetic, hybrid-kinetic and fully kinetic wave physics for space plasmas. New J. Phys. **18**(6), 065011 (2016)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



EXAMAG: Towards Exascale Simulations of the Magnetic Universe



Volker Springel, Christian Klingenberg, Rüdiger Pakmor, Thomas Guillet, and Praveen Chandrashekar

Abstract Simulations of cosmic structure formation address multi-scale, multi-physics problems of vast proportions. These calculations are presently at the forefront of today's use of supercomputers, and are important scientific drivers for the future use of exaflop computing platforms. However, continued success in this field requires the development of new numerical methods that excel in accuracy, robustness, parallel scalability, and physical fidelity to the processes relevant in galaxy and star formation. In the EXAMAG project, we have worked on improving and applying the astrophysical moving-mesh code AREPO with the goal to extend its range of applicability. We have also worked on developing new, powerful high-order discontinuous Galerkin schemes for astrophysics, on more efficient solvers for gravity, and on improvements of the accuracy of the treatment of ideal magnetohydrodynamics. In this context, we have also studied the applied mathematics required for higher-order discretization on dynamically moving meshes, thereby providing the foundations for much more efficient and accurate methods than are presently in use. Finally, we have worked towards publicly releasing two major community codes, AREPO and GADGET-4, which represent the state-of-the-art in the field.

V. Springel (✉) · R. Pakmor
Max-Planck-Institute for Astrophysics, München, Germany
e-mail: vspringel@mpa-garching.mpg.de; rpakmor@mpa-garching.mpg.de

C. Klingenberg
Universität Würzburg, Würzburg, Germany
e-mail: klingen@mathematik.uni-wuerzburg.de

T. Guillet
University of Exeter, Exeter, UK
e-mail: T.A.Guillet@exeter.ac.uk

P. Chandrashekar
TIFR Centre for Applicable Mathematics, Bengaluru, India
e-mail: praveen@tifrbng.res.in

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_12

www.dbooks.org

1 Introduction

Hydrodynamical simulations of galaxy formation have significantly matured over recent years and now enable successful predictions of the build-up of the galaxy population starting from cosmological initial conditions left behind by the Big Bang. These simulations can track the non-linearly coupled evolution of both baryons and dark matter, in principle fully accounting for their mutual influence on each other and yielding rich predictions for galaxy properties, the diffuse gas in the circumgalactic and intergalactic media, and cosmic dark matter clustering. Provided that such hydrodynamic simulations can be pushed to sufficiently large volumes, they provide the most powerful approach for forecasting non-linear cosmological observables related to clustering in different regimes and at different epochs. However, it is extremely challenging to include all the relevant physics, and to make the simulations accurate, fast, and scalable enough to be able to exploit the full capacity of today's supercomputers. The EXAMAG project has been aiming to advance novel numerical methodologies for astrophysical simulations, and to apply them right away to timely research questions in astrophysics. A particular focus has been on magnetic field predictions, and the development of higher-order methods.

In this project review, we report a subset of the results obtained within the EXAMAG project. In Sect. 2 we describe the IllustrisTNG simulations, the currently most advanced set of magnetohydrodynamical simulations of galaxy formation, as well as the Auriga simulations, which focus on predictions for our own Milky Way galaxy, in particular on the structure and origin of its magnetic field. In Sect. 3, we turn to our recent developments of discontinuous Galerkin hydro- and magnetohydrodynamics codes. We also give a short description of some of our methodological advances in combining the idea of fully dynamic, unstructured meshes with high-order discontinuous Galerkin approaches to hydrodynamics. We also report on the application of these higher order methodologies to the problem of driven isothermal turbulence, where these methods prove to be particularly powerful. We then recount in Sect. 4 some of our work on the performance and accuracy of the two cosmological hydrodynamical simulation codes GADGET-4 and AREPO, both of which we have prepared for public release to the community as part of this project. Finally, we summarize and give an outlook in Sect. 5.

2 The IllustrisTNG and Auriga Simulations

The AREPO code [31] introduced a different approach from the ones so far commonly adopted in astrophysics to evolve gas on a computer (smoothed particle hydrodynamics, SPH, and Eulerian mesh-based methods, typically utilizing adaptive mesh refinement, AMR). It employs a moving, unstructured mesh where, like AMR, the volume of space is discretized into many individual cells, but similar to SPH, these cells move with time, adapting to the flow of gas in their vicinity. As a

result, the mesh itself, constructed through a Voronoi tessellation of space, has no preferred directions or regular grid-like structure, and is highly spatially adaptive, making it ideal, in particular, for studying galaxy formation.

Our ground-breaking "Illustris" hydrodynamical calculation of galaxy formation [35] demonstrated the utility of the approach for simulations of structure formation. For the first time ever, it reproduced the observed morphological mix of galaxies and its dependence on stellar mass. Over the past years we have undertaken significant efforts to improve the underlying physics models (especially with respect to magnetic fields, and the processes regulating star formation through energetic feedback from supernovae and black holes), and the accuracy and scalability of the numerical algorithms (for example by developing a hierarchical local time-stepping algorithm for gravity). These efforts culminated in the simulation projects IllustrisTNG and Auriga.

The Next Generation (TNG) Illustris project[1] built on the technical and scientific achievements of its predecessor and pushed this line of research further. In particular, it has improved upon Illustris by including our newly developed accurate solver for ideal magnetohydrodynamics [22, 24], and by extending the dynamic range and resolution of the simulated galaxies and haloes significantly through an ambitious suite of simulations carried out on the Hazel-Hen supercomputer at the High-Performance Computing Center Stuttgart (HLRS) with the help of two large compute-time grants by the Gauss Centre for Supercomputing (GCS). We have considered three different box sizes, roughly 300, 100, and 50 Mpc on a side, and computed extensive resolution studies for each of them, yielding three series or runs, entitled TNG300, TNG100, and TNG50. The calculations used up to 24,000 cores, required 100 TB RAM, and produced 660 TB of scientific data. A visual impression of one of these simulation is given in Fig. 1. As an illustrative result, we show the clustering statistics of different matter components at the present epoch in TNG300 in Fig. 2 [33]. The calculation is able to probe deeply into the non-linear regime, over a very large dynamic range, thereby allowing predictions both for the internal structure of galaxies as well as their large-scale clustering patterns.

The scientific analysis of the IllustrisTNG simulations has started in 2018 and has already led to many important results for a vast range of scientific questions [e.g. 19, 20, 27, 33, 34], demonstrating the great utility of these methods. At the time of this writing, the TNG simulations have already produced 67 journal publications, and about 80 additional papers are currently in preparation by a network of international collaborators. In December 2018, we publicly released the data of TNG100 and TNG300 [21], with TNG50 to follow in a year's time, which will further amplify the scientific use of the simulations.

The Auriga simulations [11] use a very similar physics and numerical model as IllustrisTNG, but "zoom-in" on individual Milky Way-sized galaxies that are studied with much higher resolution. A particular focus here has been on understanding the origin of the magnetic fields in galaxies, and on predicting their present structure as

[1] http://www.tng-project.org.

www.dbooks.org

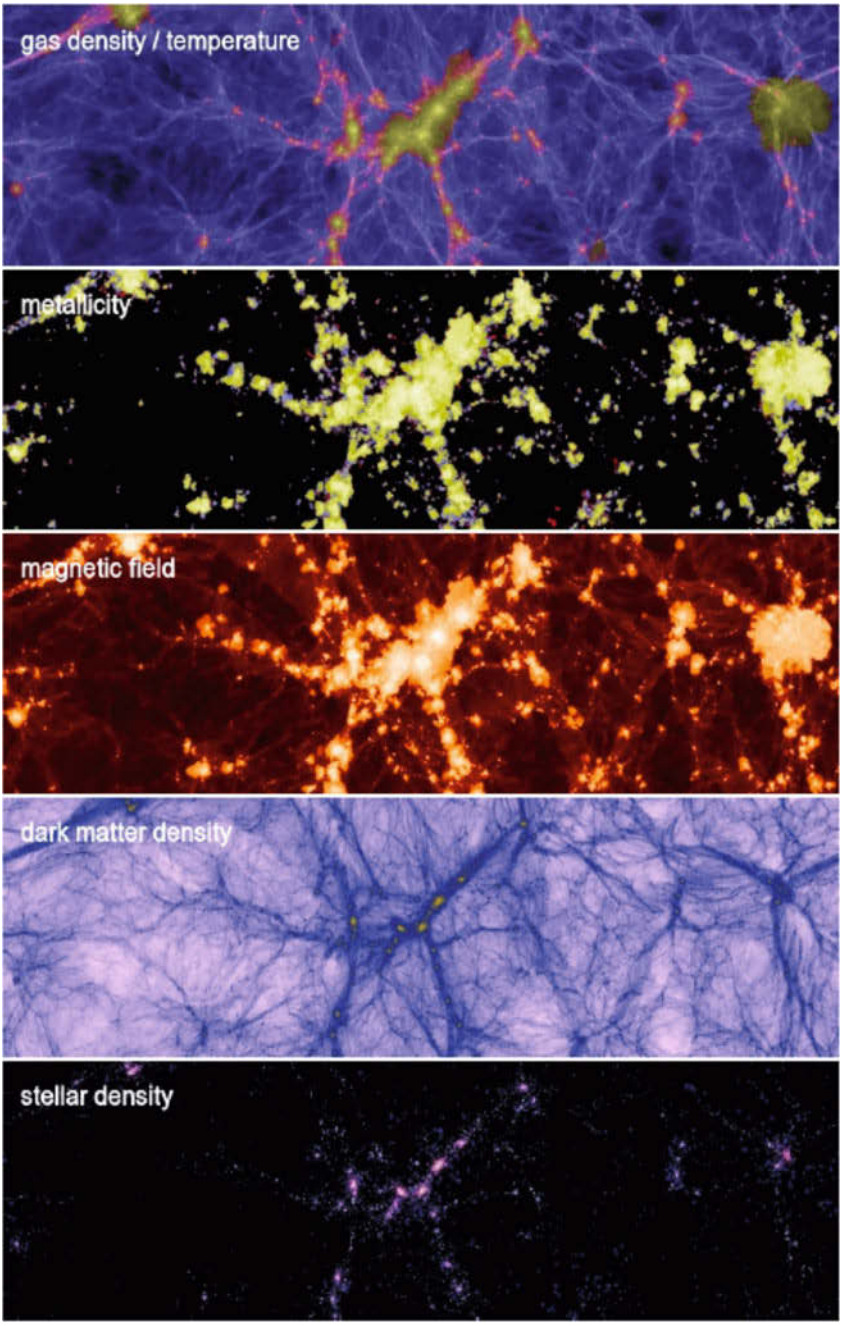


Fig. 1 Thin projections through the TNG100 simulation, showing (from top to bottom) the gas density field, the metallicity, the magnetic field strength, the dark matter density, and the stellar density

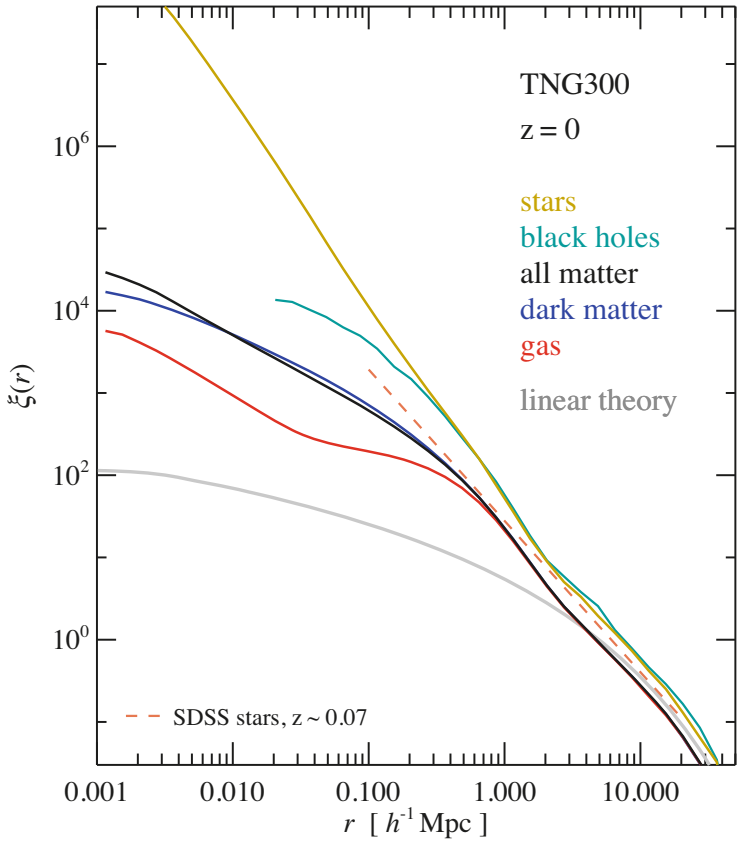


Fig. 2 The matter autocorrelation function for different mass components in our high-resolution TNG300 simulation at redshift $z = 0$, see [33]. We show results for stellar matter, gas, dark matter, black holes, and all the matter, as labelled. The linear theory correlation function is shown in grey for comparison

well as their build-up over cosmic time. In [25] we have shown that the magnetic fields grow exponentially at early times owing to a small-scale dynamo with an e-folding time of roughly 100 Myr in the centre of haloes until saturation occurs around redshift $z = 2$–3, when the magnetic energy density reaches about 10% of the turbulent energy density with a typical strength of 10–50 μG. Outside the galactic centres, differential rotation in the discs leads to linear amplification of the magnetic fields that typically saturates around $z = 0.5$–0. The final radial and vertical variations of the magnetic field strength can be well described by two joint exponential profiles, and are in good agreement with observational constraints.

We have extended the observational comparisons by computing synthetic Faraday rotation maps due to the magnetic fields [26], for different observer positions within and outside the simulated galaxies. We find that the strength of the Faraday

www.dbooks.org

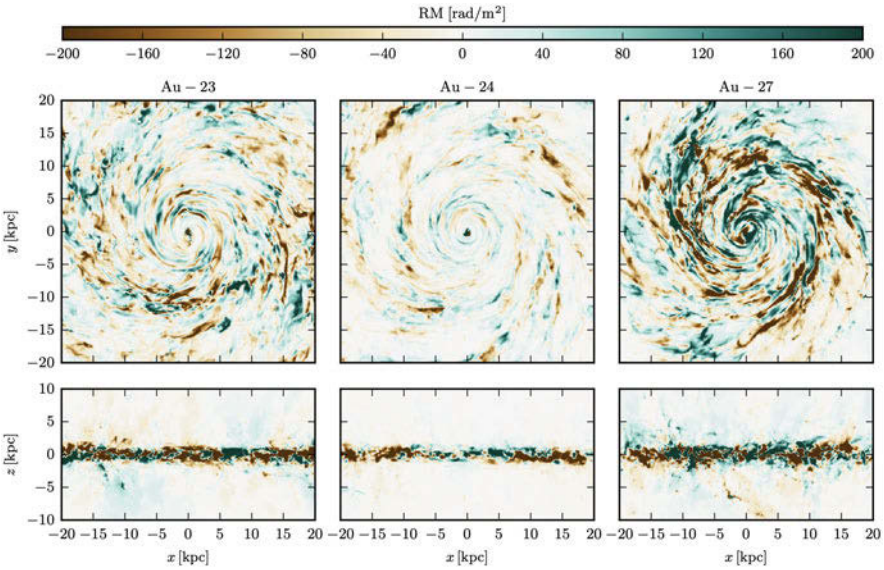


Fig. 3 Faraday rotation maps as seen by an external observer for three of the Auriga simulations. The top and bottom panels show face-on and edge-on Faraday rotation maps due to the magnetic fields in three different simulated disk galaxies Au-23, Au-24, and Au-27 [26]. These predictions compare well to observations of real galaxies such as M51

rotation of our simulated galaxies for a hypothetic observer at the solar circle is broadly consistent with the Faraday rotation seen for the Milky Way. The same holds for an observer outside the galaxy and the observed signal of the nearby spiral galaxy M51, see Fig. 3. However, we also find that the structure of the synthetic all-sky Faraday rotation maps vary strongly with azimuthal position along the solar circle. This represents a severe obstacle for attempts to reconstruct the global magnetic field of the Milky Way from Faraday rotation maps alone without including additional observables.

3 Discontinuous Galerkin Hydrodynamics for Astrophysical Applications

The AREPO code is based on a second order finite volume method on a moving mesh. An important part of the EXAMAG project was to develop a higher order method for AREPO, for which we have chosen to use a discontinuous Galerkin (DG) method due to the significant promise this class of methods holds for high performance computing. Here the hydrodynamic or magnetohydrodynamic partial differential equations are written in a weak form to construct a finite element method. On each cell, the solution and the test functions are approximated by

appropriate polynomials. Note that the approximate solution is not required to be continuous across cell boundaries. The jumps between neighboring cells are taken into account by a numerical flux function based on an approximate Riemann solver which provides stability to the method. The time discretisation is performed using a Runge-Kutta method. For more details, see [29].

Using the discontinuous Galerkin (DG) method brings with it the following advantages:

- DG works naturally on unstructured meshes which makes it suitable for adaptive meshes with hanging nodes and for moving mesh methods.
- DG can be made to work for any order of accuracy with a compact stencil and provides spectral-type accuracy.
- DG is extremely local in data communication, thus ideally suited for efficient parallelization on current computer hardware.

Due to these advantages and also the ability of DG to compute convection dominated problems in a stable and accurate manner, the EXAMAG project worked towards adopting DG methods for astrophysical applications. Our efforts in this direction followed two main paths that will be presented in the following subsections. First, in Sect. 3.1, we describe the discontinuous Galerkin method on a Cartesian mesh with automatic mesh refinement for both the Euler and magnetohydrodynamics equations. Then, in Sect. 3.2, the discontinuous Galerkin method on a moving mesh is discussed. Finally, in Sect. 3.3 we present some results on turbulence simulations with higher order numerics.

3.1 The Discontinuous Galerkin Method on a Cartesian Mesh with Automatic Mesh Refinement

In a collaboration with the EXA-DUNE project (Peter Bastian, Heidelberg) we implemented a two-dimensional hydrodynamics code in the DUNE framework with total variation bounded and positivity preserving limiters. A mesh refinement triggered by the limiting criteria was also built in, see [9].

In [10], we developed a code based on the DG method for compressible flows to incorporate and test shock indicators that can determine which cells need limiting. We showed that the choice of variables that are limited can have a major influence on accuracy; limiting the characteristic variables was compared to limiting the conserved variables, with the former being better able to control oscillations. These limiters were then combined with a shock indicator, yielding the ability to solve complex flow problems in a more efficient and accurate manner since the costly limiters need not be applied everywhere.

These investigations provided the foundations for our implementation of the discontinuous Galerkin approach in a new branch of the AREPO code called TENET, as presented in [28]. This version of the code supports adaptive mesh refinement (AMR) and is able to maintain high-order accuracy at AMR refinement

www.dbooks.org

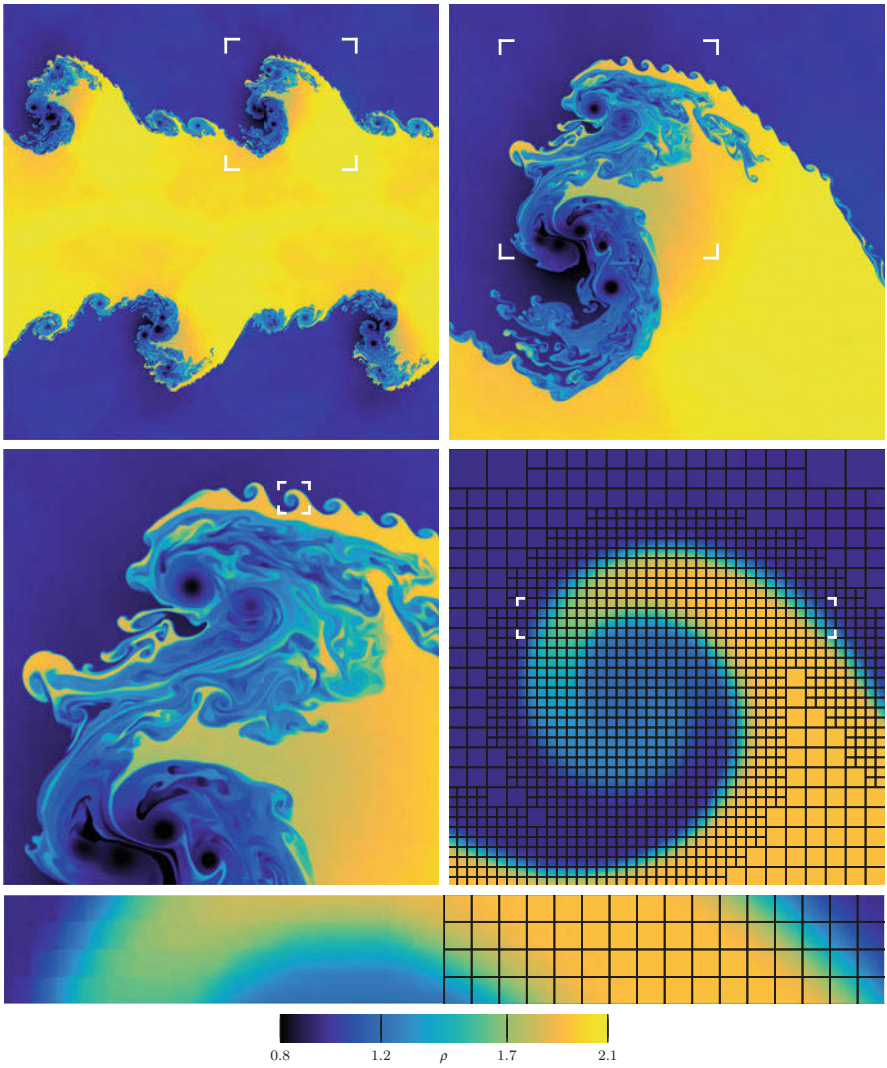


Fig. 4 A Kelvin-Helmholtz simulation with fourth order DG and adaptive mesh refinement (see [28]). The simulation starts with 64^2 cells, and refines in selected regions to an effective resolution of 4096^2. As can be seen in the bottom panel (white markers indicate regions that are enlarged in subsequent panels), the solution within every cell contains rich information, consisting of a third order polynomial

boundaries, unlike finite-volume approaches that typically fall-back to first order there. As an illustrative example, Fig. 4 depicts a simulation of a Kelvin-Helmholtz instability using TENET. In [28] we have also shown that DG has significant advantages over a finite volume method. Given the same accuracy target, a higher

order DG method requires fewer grid points than a finite volume method, allowing for much faster run times, especially for smooth solutions. The higher order DG method of TENET is also better in computing discontinuous solutions, although the numerical techniques for identifying regions where a limiter has to be applied are intricate and still not fully mature. We also showed that the DG approach automatically conserves angular momentum in smooth regions which is beneficial for many astrophysical problems involving rotating objects.

We next worked towards adding magnetic fields, leading to the model of ideal magnetohydrodynamics in fully compressible flows. The system of ideal magneto-hydrodynamical equations poses additional challenges for numerical simulations, mainly due to the need to preserve the divergence-free condition on the magnetic field, which requires specialized techniques. In [5] we developed an entropy stable finite volume scheme based on a symmetrized version of the MHD equations. A numerical flux is given which allows for the construction of an entropy conservative and entropy stable scheme. It is demonstrated how this new scheme is robust for MHD simulations due to its entropy framework, in spite of the divergence condition not being explicitly satisfied. In [6] this approach was extended to an explicit high order Runge-Kutta discontinuous Galerkin method. This methodology is then combined with techniques used to control oscillations near discontinuities, similar to [10], where these techniques were introduced for the hydrodynamical case. We assessed a different approach for the divergence constraint in [15], where post- and pre-processing methods are suggested in order to numerically maintain the divergence free constraint.

Finally, in [12] we implemented high-order MHD in the AREPO code using two different approaches for maintaining the divergence constraint, a locally divergence-free basis combined with Powell terms for stability, and a hyperbolic divergence cleaning method. Two new numerical ingredients were introduced in the DG scheme: a non-linear limiting procedure for the magnetic field, and a different discretization of the Powell terms, which was found to be a key aspect for stability and accuracy of the method. The beneficial properties of the DG method found for hydrodynamical simulations were also confirmed by Guillet et al. [12] for the MHD case. The resulting scheme shows lower advection errors and better Galilean invariance than a finite volume scheme, and hence constitutes a very promising approach for more realistic applications in an astrophysical context. In Fig. 5, we show as an example the DG simulation of a two-dimensional MHD blast wave, which is a particularly challenging test case in terms of maintaining positive solutions. In Fig. 6, we demonstrate the convergence of our DG algorithms when applied to a smooth isodensity MHD vortex problem that is advected at different angles through a periodic domain. Our numerical solutions reproduce to good accuracy the expected convergence orders for orders 2–6, both for a locally divergence-free basis and a Legendre basis. Note that for increasing order progressively more degrees of freedom per cell are needed, and thus more storage and computational work per element are required. This enlargement of the cost per element is a constant factor, however, leaving the slopes of the convergence order unaffected. The high-order methods hence always win in overall efficiency

www.dbooks.org

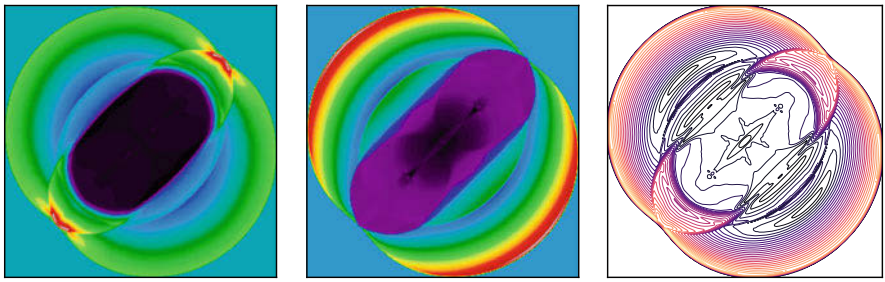


Fig. 5 A two-dimensional MHD blast wave test problem. The density, magnetic pressure and Mach number contours are shown on a 256^2 grid using a third-order discontinuous Galerkin scheme where the MHD equations are written in symmetrized form using so-called Powell terms, see [12]

and accuracy once the mesh resolution lies above a finite cross-over point (see also [12, 28] for an analysis of the accuracy of different DG-order as a function of invested CPU-time).

3.2 The Discontinuous Galerkin Method on a Moving Mesh

In a parallel endeavour, we have investigated in EXAMAG the general problem of extending higher-order methods to unstructured meshes that move along with the flow. The original AREPO approach employed a second order finite volume scheme for this purpose. As this works quite well and is able to improve the resolution of flow features by minimizing numerical dissipation from advection, we pursued extensions of this approach in three different directions.

The DG method can be proven to be entropy stable and convergent on a fixed grid for a scalar conservation law. In a first line of investigation we have shown that the entropy stability is maintained on a moving mesh. This was demonstrated in [16] and [17] for a semidiscrete arbitrary Lagrangian-Eulerian discontinuous Galerkin method. In [18] these ideas were extended to a fully discrete method. Numerical experiments have confirmed these properties also for a multi-dimensional implementation of the hydrodynamical equations.

The above studies were restricted to arbitrary Lagrangian-Eulerian discontinuous Galerkin methods, where the grid needs no remeshing. In practice, one needs to remesh the grid once in a while since otherwise the mesh quality degrades to such an extent that the computations can break down. In [1], a DG method for 2-D Euler equations was developed on triangular grids and the mesh vertices are moved in an almost Lagrangian manner. To maintain good mesh quality, only a local remeshing was performed in regions where the mesh quality has become poor. As an example, Fig. 7 shows an isentropic vortex that is also advecting. Due to the

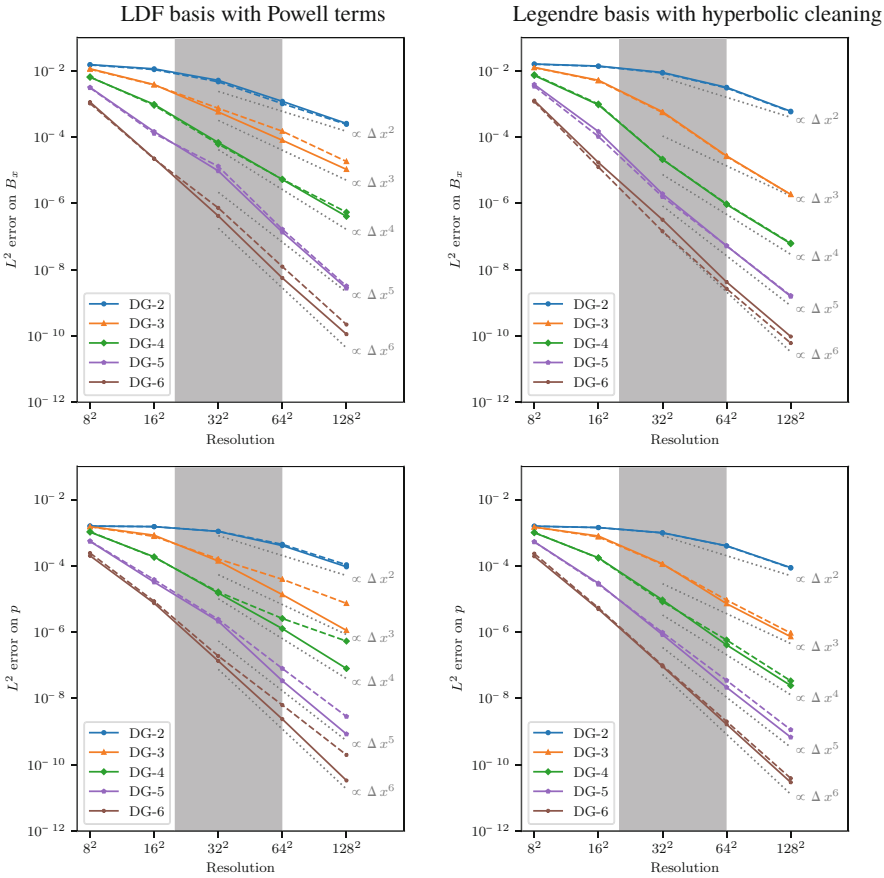


Fig. 6 Convergence of the MHD isodensity vortex problem computed for different DG orders. Solution L^2 errors are plotted for B_x (top row) and pressure (bottom row), for both a locally divergence-free (LDF) basis with Powell terms (left column), and a Legendre basis with hyperbolic cleaning (right column). Errors are measured at time $t = 20$ after the vortex has crossed the whole computational domain. Dotted lines show theoretical slopes for convergence orders 2 to 6. Solid and dashed lines correspond to errors for an advection angle $\alpha = 45°$ and $\alpha = 30°$, respectively. The shaded area corresponds to a range of resolutions for which the vortex is resolved but not over-resolved (see [12])

high shearing inside the vortex, the mesh would become heavily skewed with time, but our remeshing scheme is able to maintain good mesh quality over long time intervals.

Finally, we pursued a third direction. Would it be possible to use a moving mesh composed of Voronoi cells, as employed by AREPO for a second order finite volume method, also for a high order DG method? The difficulty here is that this type of mesh is effectively remeshed everywhere at every time step, implying that the cell connectivity and their topology can change during a timestep. This represents a

www.dbooks.org

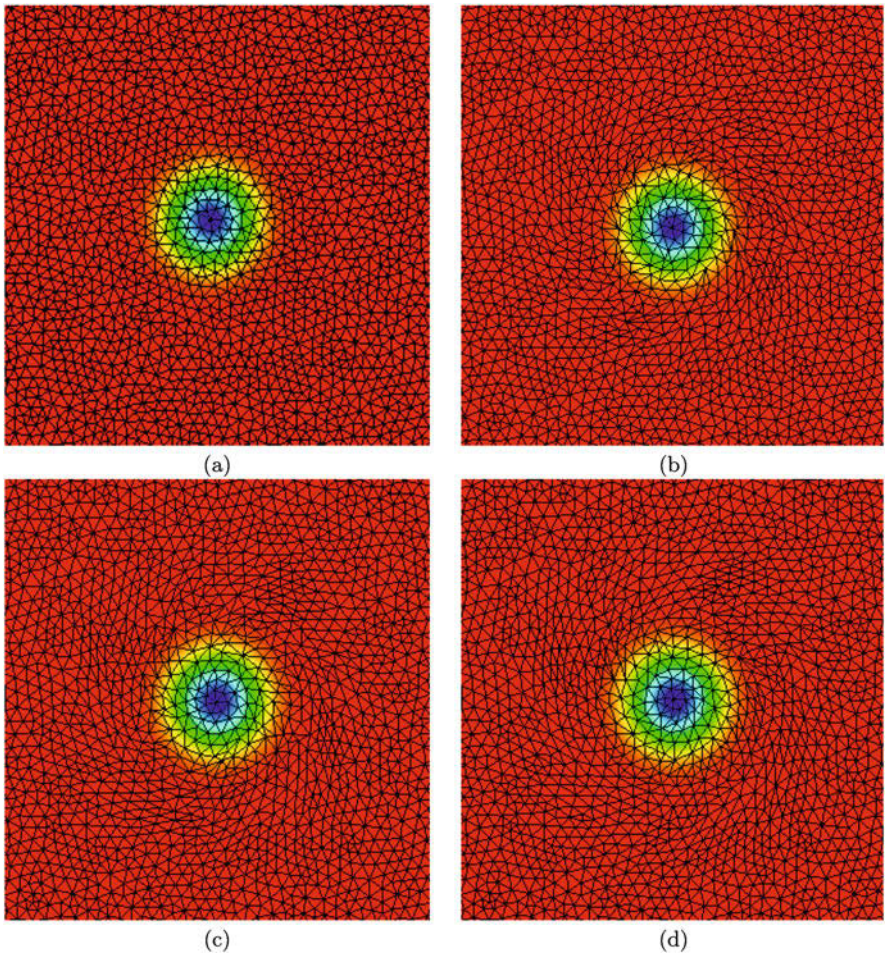


Fig. 7 A rotating isentropic vortex in two space dimensions: we show the mesh and the pressure of the solution to the Euler equations at various times: (**a**) $t = 0$, (**b**) $t = 6$, (**c**) $t = 12$, and (**d**) $t = 20$. Notice how the mesh quality does not deteriorate thanks to the local remeshing technique described in [1]

significant challenge for DG, although some guidance may also be obtained from high-order curvilinear finite element methods (e.g. [7]). Note that a higher order method is in particular high order in time, meaning it needs predictor steps between two consecutive time steps. Thus the connectivity of a cell at one time step with the cell at the next time step is needed, but a moving Voronoi cell may change the number of its sides during the step. This leads to significant geometric complications in formulating a consistent time evolution of the basis function expansion. The associated challenges were successfully overcome in [8], for the moment in two-dimensions only. But it appears conceptually straightforward to generalize this

solution to three dimensions, therefore the path to a high order DG method in 3D using the moving Voronoi mesh of AREPO is now open.

3.3 Turbulence Simulations with Higher Order Numerics

We have applied our hydrodynamic DG code developed in [28] to the simulation of three-dimensional hydrodynamic subsonic turbulence [4]. This allowed us to demonstrate that this DG implementation gives accurate results at noticeably less computational cost than a finite volume method.

Driven magnetohydrodynamical turbulence is an even richer physics problem, which is of particular importance in a variety of astrophysical contexts, including star formation in the interstellar medium, stellar atmospheres, and the X-ray emitting gas in clusters of galaxies. In Pakmor et al. (2019, in prep), we have applied our DG-MHD code developed in [12] to the problem of isothermal turbulence in a uniform box, with the goal to test different numerical schemes for preserving the MHD constraint, and for validating the effectiveness of the high-order DG approach. Of particular interest is whether there are any systematic differences between a constrained transport (CT) finite-volume MHD approach, which is able to guarantee the divergence free constraint to machine precision at all times, and the Powell and Dedner approaches for divergence control, for which we also have high order DG formulations. It is sometimes suspected that CT may be required to obtain truly accurate solutions for this problem. Reassuringly, our results, summarized in Fig. 8, do not support this view. The statistical properties of the quasi-stationary turbulent flow are very consistent between the different schemes. As expected, the growth rate of the turbulent dynamo increases with resolution and order of the scheme. While the slope of this relation is similar for all schemes, there are interesting differences in the absolute growth rate at a given resolution between the different schemes. If anything, here CT appears slightly more dissipative than the Powell approach. In contrast, the saturated magnetic energy does not seem to depend on the resolution or scheme, provided a minimum resolution is used that again depends on the scheme. Interestingly, here the Powell approach is able to numerically represent a working dynamo already at lower resolution than the CT approach.

4 Performance and Public Release of the Two Cosmological Hdyrodynamical Simulation Codes GADGET-4 and AREPO

As part of EXAMAG, we have also developed a memory efficient and fast N-body/hydrodynamical code, GADGET-4, which is primarily intended for extremely large simulations of cosmic structure formation (Gpc^3 volumes), targeting cos-

www.dbooks.org

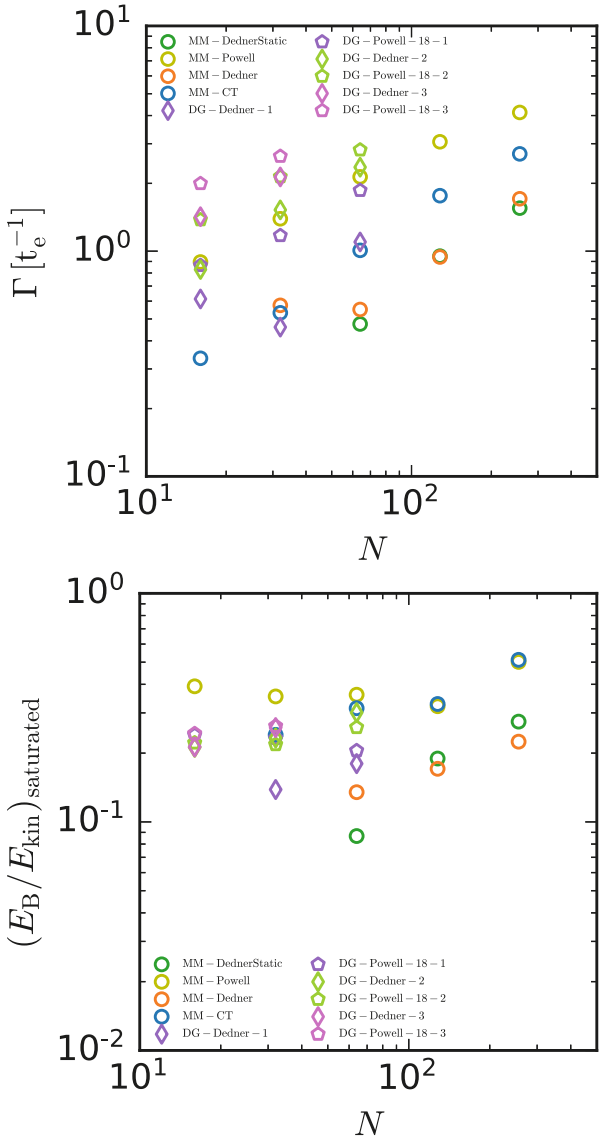


Fig. 8 Growth rate of the magnetic field strength in units of the eddy-turnover time $t_{ed} = L/(2\,\mathcal{M}\,c_s)$ (top panel) and the ratio of the magnetic to the kinetic energy in the saturated state (bottom panel) for isothermal turbulence in a uniform box for different numerical schemes and spatial resolution. N is the number of mesh cells per spatial dimension. The turbulence is driven by purely solenoidal driving and saturates at a Mach number of $\mathcal{M} \sim 0.3$. We compare different numerical schemes to control the divergence of the magnetic field, including constrained transport (CT), Dedner cleaning, and Powell terms. Also, we compare finite volume on a moving mesh (MM) with high-order DG on a Cartesian mesh

mological applications. GADGET-4 (Springel et al., 2019, in prep) represents a complete rewrite of the successful and widely used GADGET code [30], using C++ and numerous refined algorithms. For example, it supports a variety of additional gravity solvers, among them a high-order fast multipole method (FMM), as well as hierarchical local time-integration techniques.

The code is highly scalable, and can be run with two different approaches for hybrid parallelization, either a mix of MPI and OpenMP parallelization, or a novel shared memory parallelization model based on MPI-3 where one MPI rank is set aside on each shared memory node to respond to communication requests from MPI processes on remote nodes with minimum latency, thereby realizing truly one-sided communication independent of MPI progress engines. Within each node, the MPI layer can be bypassed entirely through shared-memory accesses in this method.

Figure 9 shows the effectiveness of this approach on the SuperMUC-NG machine at the Leibniz Supercomputing Centre (LRZ) in a weak scaling test, where the TreePM force computation algorithm [2, 30] is used. This approach assembles the total force as the sum of a short-range gravitational force computed with a hierarchical multipole expansion (here done with a one-sided tree algorithm, [3]), and a long-range gravity computed in Fourier space. This approach is particularly efficient for periodic cosmological simulations at high redshift where the density fluctuations are small and the large-scale residual forces nearly vanish. In this regime, pure Tree- or FMM-algorithms need to open many more nodes to accurately recover the near cancellation of most of the large-scale forces.

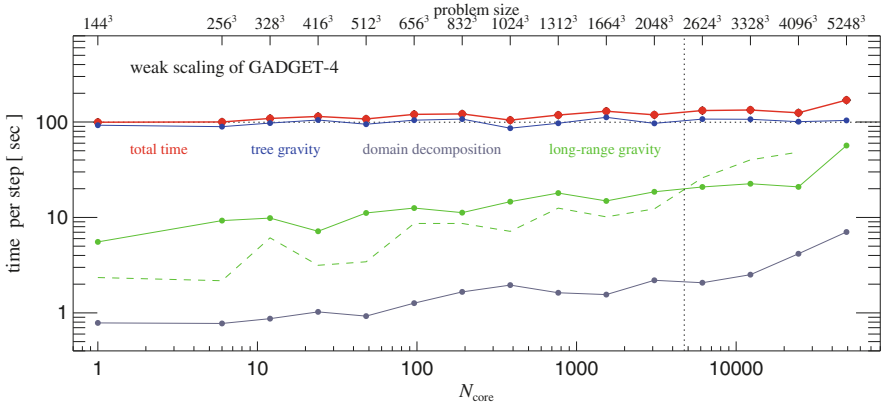


Fig. 9 Scaling of the GADGET-4 code for cosmological simulations of structure formation on the SuperMUC-NG supercomputer. The scaling of the dominant tree calculation is excellent. The long-range gravity is done through very large FFT grids, which are communication bound. Our column-based parallel FFT (solid green) scales beyond the point where the number of MPI ranks exceeds the number of mesh planes (vertical dotted line), but requires more transpose operations than a slab-based algorithm (dashed green). For the largest number of nodes, two or more islands of the machine are needed, reducing the available cross-sectional bandwidth, which impacts the scaling behavior of the communication heavy long-range gravity and domain decomposition

www.dbooks.org

The most expensive part of the calculation, the Tree-based computation of the short-range gravity scales *perfectly* to 49,152 cores (1024 nodes on SuperMUC-NG), thanks to the MPI-3 parallelization scheme, which is able to eliminate a mid-step synchronization point that was still necessary in our older conventional communication scheme for this algorithm. Because the amount of work per MPI rank stays constant in the weak scaling regime for the short-range part of the TreePM algorithm, and the amount of data that needs to be imported from neighboring ranks stays approximately constant, too, a near perfect scaling should in principle be reachable. However, because the communication pattern is complex and highly irregular, this has normally become a bottleneck for the scalability to very large numbers of MPI ranks. This is here successfully eliminated thanks to our one-sided (and fully portable) communication approach.

In contrast, the FFT-based calculation of the long-range gravity is communication-bandwidth bound and shows poorer scalability, as expected, but still stays subdominant overall. Here for the largest problem sizes an additional scaling bottleneck is resolved by GADGET-4. For a standard slab-based decomposition of the FFT (green dashed lines), there comes a point when there are more MPI ranks than mesh planes (marked by the vertical dotted line), at which point not only scalability ends, but also memory imbalance will quickly grow. This impasse is overcome in GADGET-4 with a column-based parallel FFT algorithm (green solid lines), which maintains scalability and memory balance up to the largest foreseeable problem sizes in cosmology. However, this algorithm requires twice as many transpose operations, making it more costly for small problem sizes where the slab-based approach is still viable. Like the parallel FFT, the domain decomposition algorithm is also communication bound and thus deviates from ideal scalability, but this part of GADGET-4 is fast enough to always stay subdominant. We note that the alternative hybrid parallelization through a combination of MPI and OpenMP yields very good thread scalability, see Fig. 10 (right panel), but shows slightly poorer overall scalability when the number of shared-memory nodes becomes large due to losses in its MPI communication algorithm (which still contains a midstep synchronization point).

We have also developed new on-the-fly group finding and merger-tree building techniques for GADGET-4 (which also scale well, see Fig. 10, left panel), as well as sophisticated outputting strategies for light-cones and high angular resolution maps of line-of-sight projections of various quantities, such as the total mass (for weak lensing). These features are designed to support collisionless N-body simulations with extreme particle numbers in the regime of 10^{12} particles and beyond. The huge data volume necessitates that post-processing calculations are done during the simulation as much as possible to avoid the need for enormous disk storage capacity. In fact, GADGET-4 in principle removes the need to produce any time-slices of particle data, thereby eliminating a substantial obstacle to carry out semi-analytic galaxy formation on merger trees based on simulations in the trillion particle regime. Attempting this with the same approach as in the Millennium simulation [32], where of order 100 time slices were produced and merger trees were made in

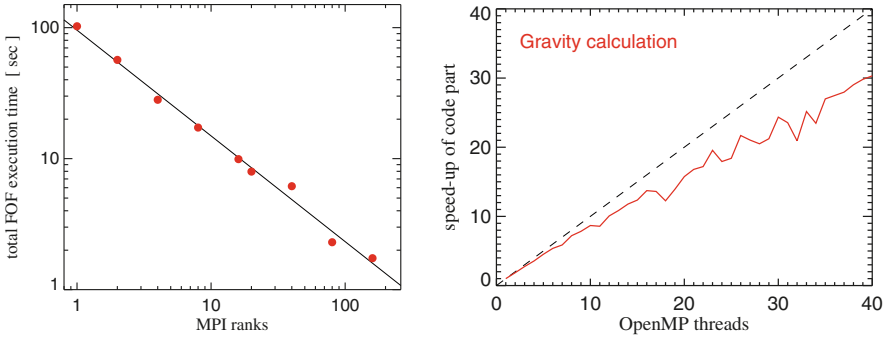


Fig. 10 The left panel shows a strong scaling test of the built-in FOF group finder in GADGET-4, while the right panels shows the speed-up of the gravity calculation in GADGET-4 as a function of the number of OpenMP threads employed when run in MPI-OpenMP hybrid mode. In the latter example, a single MPI rank on a single node of Intel Xeon 6138 cluster was used. Even though the node has two processors with 20 physical cores each, the OpenMP scaling extends well into the second processor, despite the reduced memory bandwidth this entails. In practice, GADGET-4 is best run with at least 2 MPI ranks per processor (corresponding to up to 10 OpenMP threads on this cluster). In this regime, the OpenMP scaling is excellent

post-processing, would require 6 PB of particle storage, something that we can completely avoid with the new code.

As a legacy of EXAMAG, both the GADGET-4 and AREPO codes are scheduled for public release in 2019. Weinberger et al. [36] introduces the community version of the AREPO code, and provides an overview of the available functionality as well an introduction of AREPO to new users. This is augmented with a suite of examples of different complexity, a test suite, and a support forum that is hosted on the code's website as a platform to ask questions about the code. A similar release is planned for the GADGET-4 code (Springel et al., 2019, in prep). We hope that this will foster an active and supportive user community that will contribute to the further development of these highly parallel simulation codes, preparing for their eventual use on exascale class computers, as envisioned by SPPEXA and its EXAMAG subproject.

5 Summary and Discussion

The primary research goals of the EXAMAG project have been to develop new mathematical methods and physics implementations in state-of-the-art hydrodynamical codes, allowing them to be used for groundbreaking astrophysical research that can make full use of the capabilities of current and emerging HPC platforms. Our approach consisted of leveraging a tight collaboration between applied mathematicians and numerical astrophysicists, thereby allowing a quick transfer of new mathematical ideas into applications at the forefront of today's supercomputer applications in astrophysics and cosmology.

www.dbooks.org

In hindsight, we feel that our strategy to immediately apply new numerical procedures in large application projects has been successful. This provided immediate feedback on the most promising development directions, and thus allowed us to iteratively improve codes used for production science on powerful supercomputers. Our special focus on treatments of ideal magnetohydrodynamics has allowed us to make significant progress on the physical fidelity of simulations of galaxy formation, thereby making the IllustrisTNG and Auriga projects possible in the first place. The MHD capability also provided the foundations for new solvers we developed for anisotropic diffusive transport processes, relevant especially for cosmic rays [23], thermal conduction [13] and radiative transport [14].

There is no shortage of ideas for developing the performance and capabilities of our AREPO, GADGET-4 and DG codes further in the future. Extending our DG-MHD techniques to high-order methods for self-gravity and source terms such as radiative cooling are an obvious direction. Other challenges lie in the technical aspects of parallelization, where especially the MPI-3 based shared-memory approach that we have introduced in GADGET-4 looks particularly promising for adoption in AREPO as well. It is clear that dedicated and sustained research efforts in numerical method development remain the basis for future scientific progress in computational astrophysics.

Acknowledgments The authors would like to thank the large user bases of the GADGET-2, GADGET-3, and AREPO codes for their continued use of these codes in a large number of scientific applications, thereby greatly motivating the development work carried out in the EXAMAG project. We are also grateful for computer time awarded through GCS at HLRS, LRZ and JSC.

References

1. Badwaik, J., Chandrashekar, P., Klingenberg, C.: Single-step arbitrary Lagrangian-Eulerian discontinuous Galerkin method for 1-D Euler equations. Commun. Appl. Math. Comput. (2019, submitted). Preprint arXiv:1602.09079
2. Bagla, J.S.: TreePM: a code for cosmological N-body simulations. J. Astrophys. Astron. **23**, 185–196 (2002)
3. Barnes, J., Hut, P.: A hierarchical O(N log N) force-calculation algorithm. Nature **324**(6096), 446–449 (1986)
4. Bauer, A., Schaal, K., Springel, V., Chandrashekar, P., Pakmor, R., Klingenberg, C.: Simulating turbulence using the astrophysical discontinuous Galerkin Code TENET. In: Proceedings of the SPPEXA Symposium. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2016). arXiv:1602.09079
5. Chandrashekar, P., Klingenberg, C.: Entropy stable finite volume scheme for ideal compressible MHD on 2-D Cartesian meshes. SIAM J. Numer. Anal. **54**(2), 1313–1340 (2016)
6. Chandrashekar, P., Gallego, P., Klingenberg, C.: A Runge-Kutta discontinuous Galerkin scheme for the ideal Magnetohydrodynamical model. In: Klingenberg, C., Westdickenberg, M. (eds.) Theory, Numerics and Applications of Hyperbolic Problems, Springer Proceedings in Mathematics & Statistics (PROMS), vol. 236, pp. 1203–1232 (2018)
7. Dobrev, V.A., Kolev, T.V., Rieben, R.N.: High-order curvilinear finite element methods for Lagrangian hydrodynamics. SIAM J. Sci. Comput. **34**(5), B606–B641 (2012)

8. Gaburro, E., Boscheri, W., Chiocchetti, S., Klingenberg, C., Springel, V., Dumbser, M.: High order direct Arbitrary-Lagrangian-Eulerian schemes on moving Voronoi meshes with topology changes. J. Comput. Phys. **407**, 109167 (2020). https://doi.org/10.1016/j.jcp.2019.109167
9. Gallego, P., Löbbert, J., Bastian, P., Müthing, S., Klingenberg, C., Xia, Y.: Implementing a discontinuous Galerkin method for the compressible, inviscid Euler equations in the DUNE framework. In: Proceedings in Applied Mathematics and Mechanics, vol. 14, 1 pp. 25–62 (2014)
10. Gallego, P., Klingenberg, C., Chandrashekar, P.: On limiting for higher order discontinuous Galerkin methods for 2D Euler equations. Bull. Braz. Math. Soc. (New Series) **47**(1), 335–345 (2016)
11. Grand, R.J.J., Gómez, F.A., Marinacci, F., Pakmor, R., Springel, V., Campbell, D.J.R., Frenk, C.S., Jenkins, A., White, S.D.M.: The Auriga Project: the properties and formation mechanisms of disc galaxies across cosmic time. Mon. Not. R. Astron. Soc. **467**(1), 179–207 (2017)
12. Guillet, T., Pakmor, R., Springel, V., Chandrashekar, P., Klingenberg, C.: High-order magnetohydrodynamics for astrophysics with an adaptive mesh refinement discontinuous Galerkin scheme. Mon. Not. R. Astron. Soc. **485**(3), 4209–4246 (2019)
13. Kannan, R., Springel, V., Pakmor, R., Marinacci, F., Vogelsberger, M.: Accurately simulating anisotropic thermal conduction on a moving mesh. Mon. Not. R. Astron. Soc. **458**(1), 410–424 (2016)
14. Kannan, R., Vogelsberger, M., Marinacci, F., McKinnon, R., Pakmor, R., Springel, V.: AREPO-RT: radiation hydrodynamics on a moving mesh. Mon. Not. R. Astron. Soc. **485**(1), 117–149 (2019)
15. Klingenberg, C., Pörner, F., Xia, Y.: An efficient implementation of the divergence free constraint in a discontinuous Galerkin method for magnetohydrodynamics on unstructured meshes. Commun. Comput. Phys. **21**(2), 423–442 (2017)
16. Klingenberg, C., Schnücke, G., Xia, Y.: An arbitrary Lagrangian-Eulerian local discontinuous Galerkin method for Hamilton- Jacobi equations. J. Sci. Comput. **86**(305), 1203–1232 (2017)
17. Klingenberg, C., Schnücke, G., Xia, Y.: Arbitrary Lagrangian-Eulerian discontinuous Galerkin method for conservation laws: analysis and application in one dimension. Math. Comput. **86**, 335–345 (2017)
18. Klingenberg, C., Schnücke, G., Xia, Y.: An arbitrary Lagrangian-Eulerian discontinuous Galerkin method for conservation laws: entropy stability. In: Klingenberg, C., Westdickenberg, M. (eds.) Theory, Numerics and Applications of Hyperbolic Problems, Springer Proceedings in Mathematics & Statistics (PROMS), vol. 236, pp. 1203–1232 (2018)
19. Marinacci, F., Vogelsberger, M., Pakmor, R., Torrey, P., Springel, V., Hernquist, L., Nelson, D., Weinberger, R., Pillepich, A., Naiman, J., Genel, S.: First results from the IllustrisTNG simulations: radio haloes and magnetic fields. Mon. Not. R. Astron. Soc. **480**, 5113–5139 (2018)
20. Nelson, D., Pillepich, A., Springel, V., Weinberger, R., Hernquist, L., Pakmor, R., Genel, S., Torrey, P., Vogelsberger, M., Kauffmann, G., Marinacci, F., Naiman, J.: First results from the IllustrisTNG simulations: the galaxy colour bimodality. Mon. Not. R. Astron. Soc. **475**, 624–647 (2018)
21. Nelson, D., Springel, V., Pillepich, A., Rodriguez-Gomez, V., Torrey, P., Genel, S., Vogelsberger, M., Pakmor, R., Marinacci, F., Weinberger, R., Kelley, L., Lovell, M., Diemer, B., Hernquist, L.: The IllustrisTNG simulations: public data release. Comput. Astrophys. Cosmol. **6**(1), 2 (2019)
22. Pakmor, R., Marinacci, F., Springel, V.: Magnetic fields in cosmological simulations of disk galaxies. Astrophys. J. **783**(1), L20 (2014)
23. Pakmor, R., Pfrommer, C., Simpson, C.M., Kannan, R., Springel, V.: Semi-implicit anisotropic cosmic ray transport on an unstructured moving mesh. Mon. Not. R. Astron. Soc. **462**(3), 2603–2616 (2016)
24. Pakmor, R., Springel, V., Bauer, A., Mocz, P., Munoz, D.J., Ohlmann, S.T., Schaal, K., Zhu, C.: Improving the convergence properties of the moving-mesh code AREPO. Mon. Not. R. Astron. Soc. **455**(1), 1134–1143 (2016)

www.dbooks.org

25. Pakmor, R., Gómez, F.A., Grand , R.J.J., Marinacci, F., Simpson, C.M., Springel, V., Campbell, D.J.R., Frenk, C.S., Guillet, T., Pfrommer, C., White, S.D.M.: Magnetic field formation in the Milky Way like disc galaxies of the Auriga project. Mon. Not. R. Astron. Soc. **469**(3), 3185–3199 (2017)
26. Pakmor, R., Guillet, T., Pfrommer, C., Gómez, F.A., Grand, R.J.J., Marinacci, F., Simpson, C.M., Springel, V.: Faraday rotation maps of disc galaxies. Mon. Not. R. Astron. Soc. **481**(4), 4410–4418 (2018)
27. Pillepich, A., Nelson, D., Hernquist, L., Springel, V., Pakmor, R., Torrey, P., Weinberger, R., Genel, S., Naiman, J.P., Marinacci, F., Vogelsberger, M.: First results from the IllustrisTNG simulations: the stellar mass content of groups and clusters of galaxies. Mon. Not. R. Astron. Soc. **475**, 648–675 (2018)
28. Schaal, K., Springel, V.: Shock finding on a moving mesh - I. Shock statistics in non-radiative cosmological simulations. Mon. Not. R. Astron. Soc. **446**(4), 3992–4007 (2015)
29. Shu, C.: Discontinuous Galerkin method for time-dependent problems: survey and recent developments. In: Feng, X., Karakashian, O., Xing, Y. (eds.) Recent Developments in Discontinuous Galerkin Finite Element Methods for Partial Differential Equations, pp. 25–62. Springer, Berlin (2014)
30. Springel, V.: The cosmological simulation code GADGET-2. Mon. Not. R. Astron. Soc. **364**, 1105–1134 (2005)
31. Springel, V.: E pur si muove: Galilean-invariant cosmological hydrodynamical simulations on a moving mesh. Mon. Not. R. Astron. Soc. **401**(2), 791–851 (2010)
32. Springel, V., White, S.D.M., Jenkins, A., Frenk, C.S., Yoshida, N., Gao, L., Navarro, J., Thacker, R., Croton, D., Helly, J., Peacock, J.A., Cole, S., Thomas, P., Couchman, H., Evrard, A., Colberg, J., Pearce, F.: Simulations of the formation, evolution and clustering of galaxies and quasars. Nature **435**, 629–636 (2005)
33. Springel, V., Pakmor, R., Pillepich, A., Weinberger, R., Nelson, D., Hernquist, L., Vogelsberger, M., Genel, S., Torrey, P., Marinacci, F., Naiman, J.: First results from the IllustrisTNG simulations: matter and galaxy clustering. Mon. Not. R. Astron. Soc. **475**, 676–698 (2018)
34. Torrey, P., Vogelsberger, M., Hernquist, L., McKinnon, R., Marinacci, F., Simcoe, R.A., Springel, V., Pillepich, A., Naiman, J., Pakmor, R., Weinberger, R., Nelson, D., Genel, S.: Similar star formation rate and metallicity variability time-scales drive the fundamental metallicity relation. Mon. Not. R. Astron. Soc. **477**, L16–L20 (2018)
35. Vogelsberger, M., Genel, S., Springel, V., Torrey, P., Sijacki, D., Xu, D., Snyder, G., Bird, S., Nelson, D., Hernquist, L.: Properties of galaxies reproduced by a hydrodynamic simulation. Nature **509**, 177–182 (2014)
36. Weinberger, R., Springel, V., Pakmor, R.: The Arepo public code release (2019). Preprint. arXiv:1909.04667

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



EXASTEEL: Towards a Virtual Laboratory for the Multiscale Simulation of Dual-Phase Steel Using High-Performance Computing



Axel Klawonn, Martin Lanser, Matthias Uran, Oliver Rheinbach, Stephan Köhler, Jörg Schröder, Lisa Scheunemann, Dominik Brands, Daniel Balzani, Ashutosh Gandhi, Gerhard Wellein, Markus Wittmann, Olaf Schenk, and Radim Janalík

Abstract We present a numerical two-scale simulation approach of the Nakajima test for dual-phase steel using the software package FE2TI, a highly scalable implementation of the well known homogenization method FE^2. We consider the incorporation of contact constraints using the penalty method as well as the sample sheet geometries and adequate boundary conditions. Additional software features such as a simple load step strategy and prediction of an initial value by linear extrapolation are introduced.

The macroscopic material behavior of dual-phase steel strongly depends on its microstructure and has to be incorporated for an accurate solution. For a reasonable computational effort, the concept of statistically similar representative volume

A. Klawonn (✉) · M. Lanser · M. Uran
University of Cologne, Köln, Germany
e-mail: axel.klawonn@uni-koeln.de; martin.lanser@uni-koeln.de; m.uran@uni-koeln.de

O. Rheinbach · S. Köhler
Technische Universität Bergakademie Freiberg, Freiberg, Germany
e-mail: oliver.rheinbach@math.tu-freiberg.de; Stephan.Koehler@math.tu-freiberg.de

J. Schröder · L. Scheunemann · D. Brands
Universität Duisburg-Essen, Duisburg, Germany
e-mail: j.schroeder@uni-due.de; lisa.scheunemann@uni-due.de; dominik.brands@uni-due.de

D. Balzani · A. Gandhi
Ruhr-Universität Bochum, Bochum, Germany
e-mail: daniel.balzani@rub.de; ashutosh.gandhi@rub.de

G. Wellein · M. Wittmann
Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, Germany
e-mail: gerhard.wellein@rrze.uni-erlangen.de

O. Schenk · R. Janalík
Universita della Svizzera italiana, Lugano, Switzerland
e-mail: olaf.schenk@usi.ch; radim.janalik@usi.ch

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_13

www.dbooks.org

elements (SSRVEs) is presented. Furthermore, the highly scalable nonlinear domain decomposition methods NL-FETI-DP and nonlinear BDDC are introduced and weak scaling results are shown. These methods can be used, e.g., for the solution of the microscopic problems. Additionally, some remarks on sparse direct solvers are given, especially to PARDISO. Finally, we come up with a computationally derived Forming Limit Curve (FLC).

1 Introduction

In the EXASTEEL project, we are solving challenging **nonlinear multiscale problems from computational material science** showing parallel scalability **beyond a million parallel processes.** Our software package FE2TI solves large scale **contact problems** in sheet metal forming of microheterogeneous materials and **scales to some of the largest supercomputers available today**. Although an exascale computer is not yet available, FE2TI is **exascale ready**: For our current production simulations, we have not pushed the combined parallelism of the FE^2 multiscale computational homogenization method and of our nonlinear solvers to the limit. Both, i.e., the FE^2 method by itself, as well as our nonlinear solvers are scalable to the largest supercomputers currently in production in the leading international computing facilities.[1]

In particular, as a problem, we consider the finite element simulation of sheet metal forming processes of **dual-phase (DP) steels**, whose macroscopic material behavior strongly **depends on its microscopic material properties**. A brute force discretization with respect to the microscopic structure would lead to extremely large systems of equations, which are not feasible, even on the upcoming exascale supercomputers. To give an example, a reasonable finite element discretization down to the microscopic scale would require 10^3–10^9 finite elements for a three dimensional cube with a volume of 1 μm^3. Extrapolating this to a sheet with an area of 1 m^2 and a thickness of 1 mm would lead to 10^{18}–10^{24} finite elements. A brute force simulation would also require knowledge of the complete microstructure of the steel sheet which is not available. Therefore, an efficient multiscale or homogenization approach is indispensable to save 3 to 6 orders of magnitude of

[1] In 2011, the overall scientific goal of the German DFG priority program SPP 1648 "Software for Exascale Computing" (SPPEXA) was stated as "*to master the various challenges related to [...] [the] paradigm shift from sequential or just moderately parallel to massively parallel processing*" and thereby to "*advance the frontier of parallel computing*" [4]. From the beginning, SPPEXA aimed at a true co-design, i.e., closely connecting "*computer science with the needs of Computational Science and Engineering (CSE) and HPC*" [4]. The project EXASTEEL addresses three of the main SPPEXA research areas, namely computational algorithms, application software, and programming, i.e., we have, e.g., introduced new nonlinear solver algorithms, implemented our multiscale application software FE2TI, and applied hybrid programming and performance engineering to our codes. This work was only possible in close collaboration of mathematics, computer science, and engineering.

unknowns. Our choice of a computational homogenization approach is the **FE^2 method** which is well established in engineering; see Sect. 3 for a short introduction and further references. In the FE^2 method, the microscopic and macroscopic level are discretized independently of each other. No material laws are needed for the macroscopic level, all information needed is obtained from microscopic computations based on material laws and data on the microscopic level. Let us note that the microscopic problems can be solved in parallel once the solution of the macroscopic problem is available as input. The solution of the macroscopic problem, however, requires the information of all microscopic solutions. Thus, the FE^2 method is not trivially parallelizable but requires a sequential solution of the microscopic and the macroscopic problems; this is similar to the coarse level of a hybrid two-level domain decomposition method with multiplicative coarse level and additive subdomain solvers. The nonlinear problems on both levels, the macroscopic and the microscopic one, can be solved (after linearization) using highly parallel scalable and robust implicit solvers such as parallel algebraic multigrid methods (AMG) or parallel domain decomposition preconditioners such as FETI-DP (Finite Element Tearing and Interconnecting-Dual-Primal) [27, 28, 47–50] or BDDC (Balancing Domain Decomposition by Constraints) [20, 24, 71–73] methods. These preconditioners are usually applied as part of a Newton-Krylov approach, where the tangent problem in each Newton iteration is solved using preconditioned Krylov iteration methods. A more recent approach to nonlinear implicit problems, developed extensively within EXASTEEL, is given by **nonlinear parallel domain decomposition methods**, which are applied directly to the nonlinear problem, i.e., before linearization. In such methods, the **nonlinear problem is first decomposed into concurrent nonlinear problems**, which are **then solved by (decoupled) Newton's methods in parallel**. In this project, **nonlinear FETI-DP and nonlinear BDDC domain decomposition methods** (see also Sect. 6) have been introduced and have successfully **scaled to the largest supercomputers available**—independently of the multiscale context given by the FE^2 homogenization methods, which adds an additional level of parallelism. It was found that the nonlinear domain decomposition methods can reduce communication and synchronization and thus time to solution. They can, however, also reduce the energy to solution; see Sect. 6.1.1 and [63]. These methods can be applied within our highly scalable software package FE2TI but can also be used for all problems where implicit nonlinear solvers are needed on extreme scale computers. For scaling results of the FE^2 method to more than one million MPI ranks, see Fig. 3 in Sect. 3.2 and [64]. Note that these scaling results can be obtained only using additional parallelization on the macroscopic level. Note also that our new nonlinear implicit solvers based on nonlinear FETI-DP have scaled to the complete Mira supercomputer, i.e., 7,58,000 MPI ranks (see Fig. 15 and [57]); on the JUQUEEN supercomputer [44] (see [60]) our solver based on nonlinear BDDC has scaled to 2,62,000 MPI ranks for a 3D structural mechanics problem as well as 5,24,000 MPI ranks for a 2D problem. In the present article, the software package is used to derive a virtual forming limit diagram (FLD) by simulating the Nakajima test,

www.dbooks.org

a standard procedure for the derivation of FLDs. An FLD contains a Cartesian coordinate system with major and minor strain values and a regression function of these values, which is called forming limit curve (FLC). An FLC gives the extent to which the material can be deformed by stretching, drawing or any combination of stretching and drawing without failing [77, p. v].

The software and algorithms developed here have participated in scaling workshops at the Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany (see the reports [53, 58]), as well as at the Argonne Leadership Computing Facility (ALCF), Argonne National Laboratory, USA. They have scaled on the following world-leading supercomputers in Europe, the United States, and Asia (TOP500 rank given for the time of use):

- JUQUEEN at the Jülich Supercomputing Centre, Germany; European Tier 0; TOP500 rank 9 in the year 2015 (458,752 cores; 5.8 petaflops); FE2TI and FETI-DP have scaled to the complete machine [53, 56–58, 64]; since 2015 FE2TI is a member of the High-Q Club of the highest scaling codes on JUQUEEN [53].
- JUWELS at Jülich Supercomputing Centre, Germany; European Tier 0; TOP500 rank 23 in the year 2018 (114,480 cores; 9.8 petaflops); main source of compute time for the computation of an FLD; see Sect. 5
- Mira at Argonne Leadership Computing Facility (ALCF), Argonne National Laboratory (ANL), USA; TOP500 rank 5 in the year 2015 (786,432 cores; 10.1 petaflops); FE2TI and nonlinear FETI-DP have scaled to the complete machine [54, 57]
- Theta at ALCF, USA; TOP500 rank 18 in the year 2017 (280,320 cores; 9.6 petaflops); is ANL's bridge to the upcoming first US exascale machine AURORA (or AURORA21) scheduled for 2021; BDDC domain decomposition solver has scaled to 193,600 cores [60] and recently to 262,144 cores
- Oakforest-PACS at Joint Center for Advanced High Performance Computing, Japan; TOP500 rank 6 in the year 2016 (556,104 cores; 24.9 petaflops); first deep drawing computations using FE2TI

The remainder of the article is organized as follows. In Sect. 2, we introduce the experimental test setup of the Nakajima test and the evaluation strategy of major and minor strain values described in DIN EN ISO 12004-2:2008 [77]. In Sect. 3, we briefly describe the ingredients of our highly scalable software package FE2TI, including the computational homogenization method FE^2 and the contact formulation which is integrated into the software package FE2TI since the sheet metal deformation in the Nakajima test is caused by contact. We also present some strategies to reduce computing time. In Sect. 4, we describe the numerical realization of the Nakajima test. Then, in Sect. 5, we present numerical results of several in silico Nakajima simulations with different specimens resulting in a virtual FLC. In Sect. 6, we give an overview over our highly scalable linear and nonlinear implicit solvers, including nonlinear FETI-DP and nonlinear BDDC. These methods

can be used to solve all nonlinear problems occurring in FE2TI, as shown, e.g., in [64]. In Sect. 7, performance engineering aspects regarding the sparse direct solver package PARDISO [81] are discussed. The PARDISO sparse direct solver is a central building block in our implicit solvers and in FE2TI. In Sect. 8, we introduce different improvements on the microscopic material model to even better match some experimental results.

2 Nakajima Test

Stricter CO_2 emission regulations in combination with higher passenger safety norms in the automotive industry requires steel grades with higher toughness and less weight. The class of DP steels belongs to the advanced high-strength steels and combines strength and ductility. Its favorable macroscopic properties result from the microscopic heterogeneous structure; see the beginning of Sect. 8 for further remarks.

To demonstrate the macroscopic material behavior of a specific steel grade, different material parameters and forming behaviors have to be proven. A prominent member of material characterization is the forming limit diagram (FLD). It contains major and minor strain values at failure initiation in a Cartesian coordinate system and represents the forming limits of a steel grade for one specific material thickness. In this context, material failure is already associated with the beginning of local necking in the direction of thickness and not only with crack formation [77, p. v]. The major and minor strain values vary from uniaxial to equi-biaxial tension.

The Nakajima test is a standard procedure in material testing. In the Nakajima test, a specimen is clamped between a blank holder and a die and a hemispherical punch is driven into the specimen until a crack can be observed; see Fig. 1 (left). Friction between the forming tool and the specimen has to be avoided as much as possible. Therefore, different lubrication systems can be applied; see [77, Ch. 4.3.3.3]. To get different pairs of major and minor strains, one has to use at least five different shapes of sample geometries and for each shape, one has to carry out three different valid tests [77]. The recommended shapes of the sample sheet geometries are described in [77, Ch. 4.1.2], see also Sect. 4.1 and Fig. 1 (right) for an example of a permissible sample sheet. In experiments, the surface of a specimen is equipped with a regular grid or a stochastic pattern and is recorded by one or more cameras during the deformation process.

There are at least two different strategies to get the pair of major and minor strains for the FLC, namely the cross section method [77] and a method based on thinning rates proposed by W. Volk and P. Hora [97]. Since the FLC gives information about material deformation without failing, we are interested in major and minor strains just before localized necking occurs.

www.dbooks.org

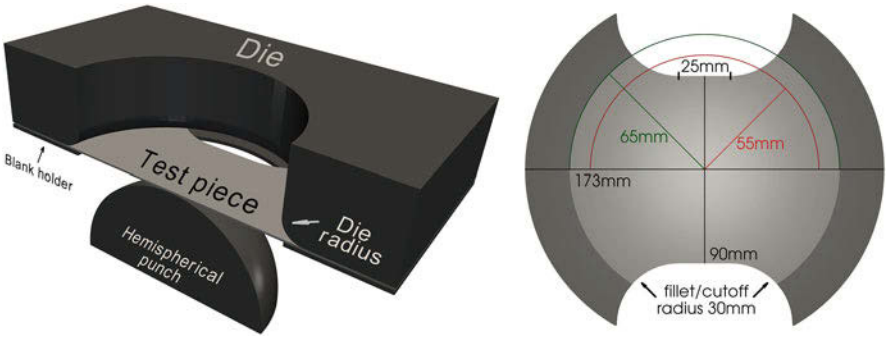


Fig. 1 **Left:** Cross section of the initial test setup of the Nakajima test. **Right:** Dimensions of a specimen used for the simulation of the Nakajima test with a shaft length of 25 mm and a parallel shaft width of 90 mm. The inner (red) circle represents the inner wall of the die and the outer (green) circle represents the beginning of the clamped part between die and blank holder. Material outside the outer (green) circle is only considered for a width of the parallel shaft of 90 mm or more (dark grey)

In the method based on thinning rates, the last recorded image before localized necking occurs is explicitly determined. This specific image is used to derive major and minor strains for the FLC.

The **cross section method** is standardized in DIN EN ISO 12004-2:2008 [77]. It uses knowledge about the position of the crack and evaluates major and minor strain values in the last recorded image before crack along cross sections perpendicular to the crack. Then, from these values, the state immediately before material failure is interpolated. Cross sections have a length of at least 20 mm at both sides of the crack. One cross section cuts exactly through the center of the crack and one or two cross sections are positioned above and below with a distance of about 2 mm. For each cross section, we want to compute a pair of major and minor strains $\overline{\varepsilon}_1^{\mathrm{FLC}}$ and $\overline{\varepsilon}_2^{\mathrm{FLC}}$, which represent the major and minor strains just before the beginning of plastic instability.[2] Therefore, we have to fit an inverse second-order polynomial using a least squares fit; see Figs. 2 and 8 (bottom). Instead of fitting inverse second-order polynomials to the values along the cross sections we fit second order polynomials to the inverse of the values. For the least squares fit we have to determine optimal fit windows for both sides of the crack separately; see Figs. 2 and 8 (bottom). For a detailed description of the procedure we refer to [77]. Let us note that $\overline{\varepsilon}_1^{\mathrm{FLC}}$ and $\overline{\varepsilon}_2^{\mathrm{FLC}}$ in general never exist during the deformation process. Hence, these numbers do not have a physical meaning [97].

[2]Note that here and in the following, all macroscopic variables and objects are denoted with an overline to distinguish them from microscopic variables and objects.

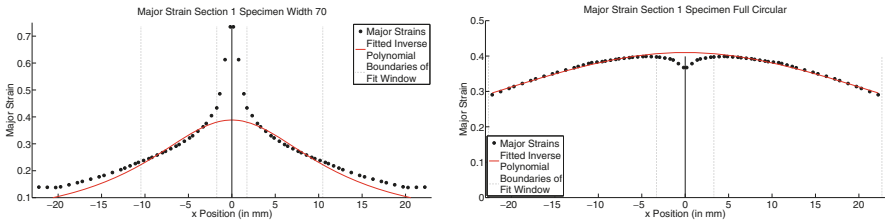


Fig. 2 Fitted inverse second-order polynomials to the major strain values along the first cross section just before material failure. See also the description of the cross section method in Sect. 2. Optimal fit windows are computed as described in [77, Ch. 5.2.3, 5.2.4]. **Left:** Specimen with a width of the parallel shaft of 70 mm. **Right:** Full circular specimen

3 FE2TI: A Highly Scalable Implementation of the FE^2 Algorithm

For the finite element simulation of the Nakajima test, we use our FE2TI software package [9, 52, 57, 64], which is a C/C++ implementation of the FE^2 computational homogenization approach [29–31, 33, 70, 75, 86, 87, 91]. It is based on PETSc [6] and MPI. The multiscale simulations based on FE2TI and using FETI-DP and BoomerAMG as solvers are a "CSE Success Story" in SIAM Review [80, p. 736].

3.1 FE^2 Algorithm

For DP steel, the overall macroscopic material behavior strongly depends on its microscopic properties. Assuming that the macroscopic length scale $\overline{L}$ is much larger than the length scale l representing the microscopic heterogeneities, i.e., $\overline{L} \gg l$, the scale separation assumption is satisfied and a scale bridging or homogenization approach such as the FE^2 method can be applied.

The idea of the FE^2 approach is to discretize the micro- and macroscopic scale separately using finite elements. The macroscopic domain is discretized without any consideration of the microscopic material properties, i.e., the material is assumed to be homogeneous from a macroscopic point of view. Additionally, a microscopic boundary value problem is defined on a representative volume element (RVE) which is assumed to represent the microscopic heterogeneities sufficiently. One microscopic finite element problem is assigned to each macroscopic Gauß point and the phenomenological law on the macroscopic level is replaced by volumetric averages of stresses and associated consistent tangent moduli of the microscopic solution. Note that the boundary values of the microscopic level are induced through the macroscopic deformation gradient at the integration point the microscopic problem is attached to.

www.dbooks.org

To derive an RVE representing a realistic microstructure, electron backscatter diffraction is used; see [14]. Note that for DP steel the martensitic inclusions in the ferrite are quite small and widely spread, which enforces a fine discretization to incorporate the heterogeneities sufficiently. To overcome this problem, we make use of so called statistically similar RVEs (SSRVEs) [8, 83], which are constructed in an optimization process with only inclusions of simple geometry such as ellipsoids, but describe the mechanical behavior in an approximate way. Note that the constructed ellipsoids are simpler than the realistic microstructure and hence, the SSRVE can be discretized with a coarser grid.

For further details such as the derivation of consistent tangent moduli we refer to the literature [33, 87] and to earlier works on computational homogenization in the EXASTEEL project [9, 52, 57, 64].

3.2 FE2TI Software Package

The FE2TI software package was developed within the EXASTEEL project and has been successfully used for the simulation of tension tests of DP steel [9, 52, 57, 64]. It belongs to the highest scaling codes on the European Tier-0-supercomputer JUQUEEN since 2015.[3]

For comparably small sizes of microscopic problems, we can solve the resulting tangent problems with a sparse direct solver such as PARDISO [81], UMFPACK [22], or MUMPS [2]. For larger sizes of microscopic problems, we have to use efficient parallel solvers which are also robust for heterogeneous problems. Such methods are Newton-Krylov methods with appropriate preconditioners such as domain decomposition or (algebraic) multigrid or nonlinear domain decomposition methods, possibly, combined with algebraic multigrid.

In our software package, Newton-Krylov-FETI-DP and the more recent highly scalable nonlinear FETI-DP methods, which were developed in this project (see [51, 54, 59] and Sect. 6.1), are integrated. Other nonlinear domain decomposition approaches are the related Nonlinear FETI-1 or Neumann-Neumann approaches [13, 78] or ASPIN [17, 18, 35–37, 40, 41, 66]. Furthermore, FE2TI can also use the highly scalable algebraic multigrid implementation BoomerAMG [5, 38] from the hypre package [25] for micro- as well as macroscopic problems. The scalability of BoomerAMG was recently improved for problems in elasticity, and scalability of BoomerAMG to half a million ranks was then achieved within the EXASTEEL project [5] in close collaboration with the authors of BoomerAMG.

For the contact simulations presented here, we consider problem sizes for which we can use the direct solver package PARDISO to solve the resulting tangent problems on the microscopic as well as on the macroscopic level. This limits the

[3] https://www.fz-juelich.de/ias/jsc/EN/Expertise/High-Q-Club/FE2TI/_node.html and also https://juser.fz-juelich.de/record/188191.

size of our problems but is suitable for mid-sized supercomputers. In our opinion, this is a relevant criterion for the applicability in industry. Using our parallel nonlinear solvers, the FE2TI package scales up to the largest machines even without making use of the full scaling potential of the solvers (see Fig. 3); and for the combination of large macroscopic problems with large RVEs an exascale computer will be necessary in the future. While Fig. 3 (left) represents a weak scaling study with large RVEs and comparably small macroscopic problems, in Fig. 3 (right) the macroscopic problems are larger. Therefore, in the latter case, a parallel macroscopic solve using GMRES with an AMG preconditioner is beneficial. The scalability in Fig. 3 (right) somewhat suffers from an increase in the numbers of Newton iterations. Let us remark that the setup in Fig. 3 (right) is the setup of a typical production run. The strong scaling potential of FE2TI is also presented in Fig. 4; see [9] for details. For more scalability results on different architectures also see [64].

Even if the macroscopic problem is solved with a direct solver, the assembly process is parallelized. For the incorporation of a material law on the microscopic level the software is equipped with an interface to FEAP, and we use an implementation of a J_2 elasto-plasticity model [65]. Material parameters are chosen as in Brands et al. [14, Fig. 10].

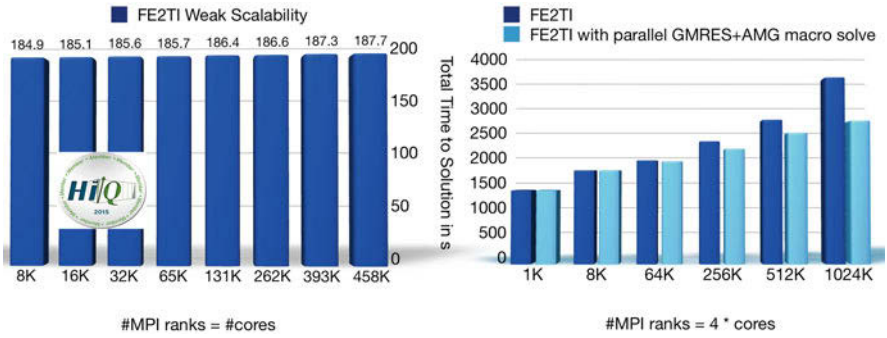


Fig. 3 Weak scalability of the FE2TI software on the JUQUEEN supercomputer [44]. **Left:** Time to solution of a single load step solving a three-dimensional heterogeneous hyperelastic model problem; uses Q1 finite elements (macro) and P2 finite elements (micro); 1.6M d.o.f. on each RVE; 512 FETI-DP subdomains for each RVE; the macroscopic problem size grows proportionally to the number of MPI ranks while the microscopic problem size is fixed; corresponding data in [57, Tab. 2]; High-Q club computation in 2015. **Right:** Total time to solution for 13 load steps solving 3D heterogeneous plasticity; uses Q1 finite elements (macro) and P2 finite elements (micro); 200K d.o.f. on each RVE; 64 FETI-DP subdomains for each RVE; the macroscopic problem is increased proportionally to the number of MPI ranks; for the larger problems using parallel AMG for the problem on the macroscale, instead of a sparse direct solver, is beneficial; see also [64, Fig. 15]

www.dbooks.org

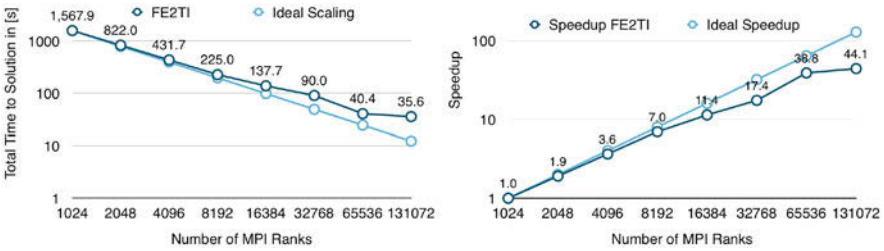


Fig. 4 Strong scalability of the FE2TI software for a nonlinear elasticity problem. Macroscopic problem with 32 finite elements; each RVE with 107K degrees of freedom is solved using 512 FETI-DP subdomains. Simulation of one macroscopic load step. **Left:** Total time to solution; **Right:** Speedup. Figures from [9]

3.3 Contact Kinematics and Incorporation of Contact Constraints for Frictionless Contact in FE2TI

For the simulation of the Nakajima test, we have to consider contact between the deformable specimen $\overline{\mathcal{B}}$ and different rigid tools $\overline{\mathcal{T}}_i$, $i = 1, 2, 3$, namely the hemispherical punch, blank holder, and die; see Fig. 1 (left). Therefore, we implemented a contact algorithm on the macroscopic scale in FE2TI. To simplify the notation, we consider an arbitrary rigid tool $\overline{\mathcal{T}}$ in the following.

A general convention in contact formulations is to consider one contact partner as the master body and one contact partner as the slave body [68, 99].

Only points of the contact surface of the slave body are allowed to penetrate into the master body. Following [68], one can choose the rigid surface as slave surface as well as a master surface, and in [99, Rem. 4.2] it is recommended to use the rigid surface as master surface; we have applied the latter in our simulations. Nevertheless, the contact contributions to the stiffness matrix and the right-hand-side are computed in the coordinate system of the deformable body.

In every iteration, we have to check for all finite element nodes $\overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_{\overline{\mathcal{B}}}$ of the contact surface of $\overline{\mathcal{B}}$ whether it penetrates into $\overline{\mathcal{T}}$ or not; see Fig. 5 for a simplified illustration. For each $\overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_{\overline{\mathcal{B}}}$ we have to determine the related minimum distance point $\overline{x}_{\overline{\mathcal{T}}}^{\min} := \min_{\overline{x}_{\overline{\mathcal{T}}} \in \overline{\Gamma}_{\overline{\mathcal{T}}}} ||\overline{x}_{\overline{\mathcal{B}}} - \overline{x}_{\overline{\mathcal{T}}}||$ of the contact surface of $\overline{\mathcal{T}}$. Now, we can formulate a non-penetration condition

$$\overline{g}_{NP} = (\overline{x}_{\overline{\mathcal{B}}} - \overline{x}_{\overline{\mathcal{T}}}^{\min}) \cdot \overline{n}_{\overline{\mathcal{T}}}^{\min} \geq 0, \ \overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_{\overline{\mathcal{B}}}. \tag{1}$$

Alternatively, for all points $\overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_c := \left\{ \overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_{\overline{\mathcal{B}}} \mid \overline{g}_{NP} < 0 \right\}$ which penetrate into the master body, the amount of penetration can be computed by

$$\overline{g}_N = (\overline{x}_{\overline{\mathcal{B}}} - \overline{x}_{\overline{\mathcal{T}}}^{\min}) \cdot \overline{n}_{\overline{\mathcal{T}}}^{\min}, \ \overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_c, \tag{2}$$

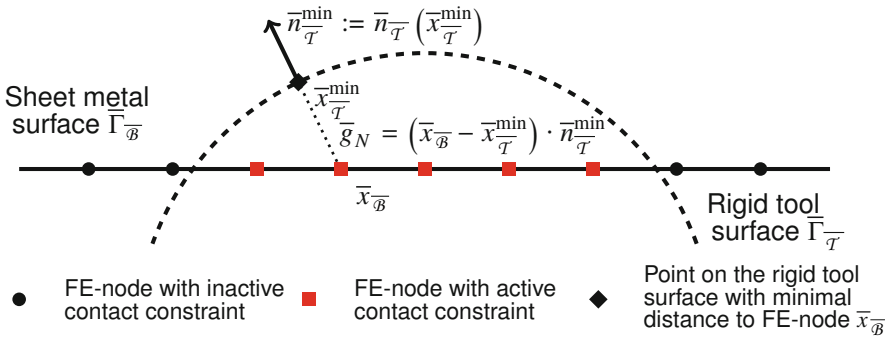


Fig. 5 Illustration for the determination of active contact nodes and the amount of penetration

and is set to zero for all other points. Here, $\overline{n}_{\overline{\mathcal{T}}}^{\min}$ is the outward normal of the rigid tool at $\overline{x}_{\overline{\mathcal{T}}}^{\min}$; see Fig. 5.

Since the contact partners of the sheet metal are assumed to be rigid, the tools are not discretized by finite elements but the contact surfaces are characterized by analytical functions. This also simplifies the computation of the related minimum distance point and, hence, the computation of the outward normal direction as well as of the amount of penetration. For a detailed description of contact between two deformable bodies we refer to [99, Ch. 4.1].

As in standard finite element simulations of continuum mechanics problems, we are interested in the minimization of an energy functional $\widetilde{\Pi}$, but under additional consideration of an inequality constraint due to the non-penetration condition (Eq. (1)). Constrained optimization can be performed, e.g., using the (quadratic) penalty method, where an additional penalty term is added to the objective function if the constraint is violated; see [76, Ch. 17]. Let us note that the incorporation of contact constraints into a finite element formulation does not change the equations describing the behavior of the bodies coming into contact [99].

There are several different strategies to incorporate contact constraints into finite element formulations, where the penalty method and the Lagrange multiplier method are the most prominent members; see [99, Ch. 6.3]. The penalty method is the most widely used strategy to incorporate contact constraints into finite element simulation software [99, Ch. 10.3.3] because the number of unknowns does not increase. In comparison to the Lagrange multiplier method [99], the contact constraints are only resolved approximately and a small penetration depending on the choice of the penalty parameter $\overline{\varepsilon}_N > 0$ is allowed. For $\overline{\varepsilon}_N \to \infty$, the exact solution of the Lagrange multiplier method is reached, but for higher penalty parameters $\overline{\varepsilon}_N$, the resulting system of equations becomes ill-conditioned [99]. For a suggestion of a choice of the penalty parameter $\overline{\varepsilon}_N$, see [99, Remark 10.2].

www.dbooks.org

Using the penalty method, we have to add an additional term

$$\widetilde{\Pi}_P = \int_{\overline{\Gamma}_c} \frac{1}{2} \cdot \overline{\varepsilon}_N \cdot \overline{g}_N^2 \, \mathrm{dA}$$

to the energy functional $\widetilde{\Pi}$ for all active contact nodes $\overline{x}_{\overline{\mathcal{B}}} \in \overline{\Gamma}_c$ [99, Ch. 6.3]. The penalty parameter $\overline{\varepsilon}_N$ can be interpreted as the stiffness of a spring in the contact interface; see [99, Ch. 2.1.3]. Let us note that the definition of an active set is different from standard textbooks as [76, Def. 12.1], where points belong to the active set if they fulfill equality of the inequality constraint. Other authors like Konyukhov and Schweizerhof considered the Heaviside function to follow the common definition of an active set; see, e.g., [67, 68]. Since the energy functional is changed due to the contact constraints, also the resulting stiffness matrix and right-hand-side are affected.

3.4 Algorithmic Improvements in FE2TI

In simulations making use of load stepping (or pseudo time stepping) as a globalization strategy, as is the case in FE2TI (see Sect. 4), the time to solution strongly depends on the number of load steps as well as on the number of macroscopic Newton iterations per load step. The required time of a single macroscopic Newton step again depends on the time to solution of the microscopic problems.

While a large load step may seem desirable, it can lead to slow convergence or even divergence; convergence can be forced by reducing the load step size thus increasing the total number of load steps; this can be observed in Table 1. To adapt the size of the load steps, we use a simple dynamic load step strategy; see Sect. 3.4.1.

Keeping the number of macroscopic Newton iterations as small as possible is directly related to a good choice of the initial value of a single load step. For a better prediction of the initial value, we use linear extrapolation; see Sect. 3.4.2. This is especially relevant for our contact problems where quadratic convergence of Newton's method can be lost.

Table 1 First 2 mm displacement of the forming tool with constant load increments $\overline{l}$ of 3.125e−3 (first row) and 1.0e−1 (second row) for the specimen with a parallel shaft width of 50 mm; two MPI ranks per core; computed on JUWELS [45]

	Cov. dist. punch	Load steps	Newton its.	∅ Newton its. per load step	Runtime	∅ Time per load step	∅ Time per Newton it.
Load increment 3.125e−3	2 mm	640	966	1.51	7595.30 s	11.87 s	7.86 s
Load increment 1.0e−1	2 mm	20	130	6.50	1407.03 s	70.45 s	10.82 s

3.4.1 Dynamic Loadstepping

Our load step strategy depends on macroscopic as well as microscopic information. The macro load increment $\bar{l}$ is reduced when microscopic stagnation is observed or when a maximum number of macroscopic Newton iterations per load step is reached. Stagnation on the RVE level is detected whenever the norm of the current microscopic Newton iteration compared to the previous one does not reduce after more than six microscopic Newton iterations or if the number of microscopic Newton iterations is larger than 20. The load step is increased based on the number of macroscopic Newton iterations per load step. Note that $\bar{l}$ is not allowed to exceed a predefined maximum load increment $\bar{l}^{\max}$. For details, see Fig. 6.

Whenever stagnation in a microscopic problem occurs, the microscopic solver gives this information to the macroscopic level and the load step is repeated with a reduced load increment. Otherwise, stagnation of a microscopic problem would lead to a simulation break down due to a missing tangent modulus and stresses in the macro Gauß point where the micro problem is attached.

3.4.2 (Linear) Extrapolation

For Newton-type methods, it is important to choose a good initial value. If the initial value is close to the solution, only a few Newton iterations are necessary. As in [64],

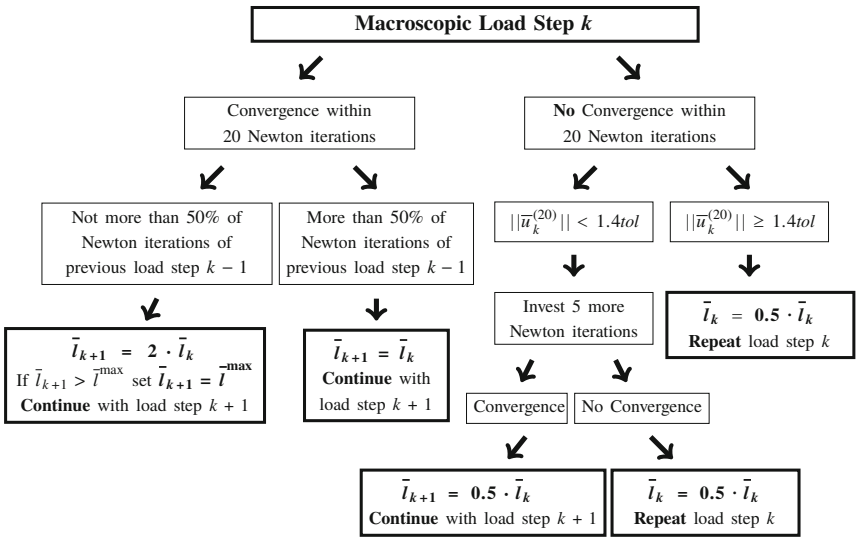


Fig. 6 Schematic procedure of reduction and increase of the load increment $\bar{l}$ depending on macroscopic events. Here, $\bar{l}^{\max}$ is a predefined maximum load, $||\bar{u}_k^{(20)}||$ is the norm of the solution of the 20th Newton iteration of the current load step k, and tol is the Newton tolerance

www.dbooks.org

we use linear extrapolation based on the converged solutions $\overline{u}_k$ and $\overline{u}_{k-1}$ of the former two load steps to guess a good initial value $\overline{u}_{k+1}^{(0)}$,

$$\overline{u}_{k+1}^{(0)} = \overline{u}_{k-1} + \frac{\overline{l}_{k+1} - \overline{l}_{k-1}}{\overline{l}_k - \overline{l}_{k-1}} \cdot (\overline{u}_k - \overline{u}_{k-1}).$$

In [64], this technique was already successfully used in the FE^2 simulations using the FE2TI software package without contact constraints. For results using second order extrapolation, we refer to [95, Ch. 4.2.2].

3.4.3 Checkpoint/Restart

To perform the simulation of the Nakajima test until material failure of a sample sheet, i.e., until a failure zone occurs in the metal sheet, often significant computation times are needed, even if the full supercomputer is available. To reduce the consequences of hardware failures and also to overcome specific wall time limits on HPC systems, we equipped our FE2TI package with synchronous Checkpoint/Restart (CR). We integrated the CRAFT library (**C**heckpoint/**R**estart and **A**utomatic **F**ault **T**olerance) [88], which was developed in the second phase of the SPPEXA project ESSEX. Let us note that we use synchronous application level checkpointing with different checkpoints for the macroscopic level and the microscopic level.

In [21], different checkpoint intervals are introduced based on the expected run time of the simulation and the mean time of hardware failure of the HPC system the software is performed on, but in all our simulations presented here, we choose a fixed checkpoint interval of 75 load steps. Here, we do not take into account that the load increment may change during the simulation and that small load increments are usually faster. As an improvement, the checkpointing could take into account the displacement of the forming tool or a fixed wall time interval could be used which also could depend on the mean time of hardware failure.

4 Numerical Simulation of the Nakajima Test Using FE2TI

For the simulation of the Nakajima test, we use our highly scalable parallel software package FE2TI; see Sect. 3. For the solution of the boundary value problems on both scales, we here use the sparse direct solver package PARDISO [81]. Since we are considering a DP600 grade of steel, we use a fitted J_2 elasto-plasticity model on the microscopic level; see [14, Fig. 10]. Throughout this article, we use structured Q2 finite element discretizations for the sample sheet and an unstructured P2 finite element discretization for the RVEs. Both, the macroscopic as well as the

microscopic meshes, are generated using the open source software package GMSH [34]. We use the load stepping and extrapolation described in Sects. 3.4.1 and 3.4.2.

In the Nakajima test, the macroscopic deformation is driven by the rigid punch. Hence, load stepping is applied to the movement of the forming tool, where the hemispherical punch moves a small step in upward direction in each load step.

Since in reality a tribological system is set up to avoid friction between the hemispherical punch and the sheet metal [77], we consider frictionless contact. Hence, we have to deal exclusively with contact conditions in normal direction of the rigid tools. Considering frictionless contact, we neglect friction between the specimen and the blank holder or die.

4.1 Description of Specimen Geometry

In our simulations, we consider specimens with a central parallel shaft and also a completely circular specimen. The length of the parallel shaft is 25 mm and the fillet radius is 30 mm, which both fit to the normed range in [77]; also see Fig. 1 (right).

For all specimens, the material is assumed to be completely clamped by the bead, which has a radius of 86.5 mm. We therefore only consider material points $\overline{p} = \left(\overline{p}_x, \overline{p}_y, \overline{p}_z\right)$ which have a distance of at most 86.5 mm to the center of the sample sheet; see Fig. 1 (right) for an example of a sample sheet with a parallel shaft width of 90 mm. In our simulations, the sample sheet is always 1 mm thick, and we consider specimens with a parallel shaft width of 30, 50, 70, 90, 100, 110, 125, and 129 mm as well as the completely circular sample sheet. Note that the center $\overline{c}^b = (\overline{c}_x, \overline{c}_y, \overline{c}_z^b)$ of the bottom surface of all sample sheets is placed in the origin of the coordinate system.

The specifications of the rigid tools are also within the range given in [77]. The radius of the hemispherical punch is 50 mm. The blank holder is a square plate of 173 mm $\times$ 173 mm with a circular hole in the middle with a radius of 55 mm; see the inner circle in Fig. 1 (right). The die is placed within a distance of 5 mm to the rigid punch, i.e., the inner wall of the die also has a radius of 55 mm; see, again, the inner circle in Fig. 1 (right). The die radius (see Fig. 1 (left)) is 10 mm, i.e., all material points $\overline{p}$ with $\sqrt{\overline{p}_x^2 + \overline{p}_y^2} \geq 65$ are potentially clamped; see the outer circle in Fig. 1 (right).

For all sample sheets with a parallel shaft width less than 90 mm as well as for the completely circular specimen, we only consider material points $\overline{p}$ with $\sqrt{\overline{p}_x^2 + \overline{p}_y^2} \leq 65$ and choose all points with $\sqrt{\overline{p}_x^2 + \overline{p}_y^2} = 65$ as Dirichlet boundary nodes. For specimens with wider parallel shaft widths, we choose boundary conditions analogously to [43]; see also [95].

www.dbooks.org

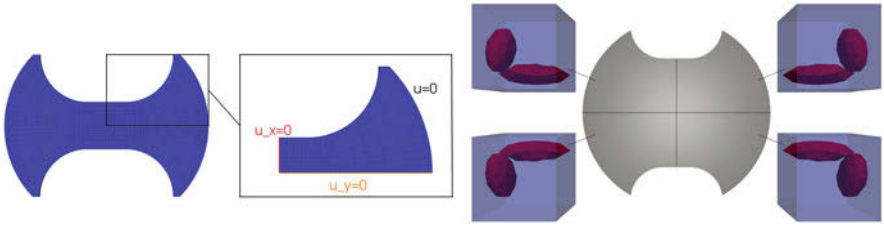


Fig. 7 **Left:** Symmetric computational domain from the full sample sheet and additional Dirichlet boundary conditions along the symmetric boundary surfaces. **Right:** Orientation of the RVEs for the different quadrants of the full geometry of the sample sheet after mirroring the first quadrant due to the symmetric assumption

4.2 Exploiting Symmetry

The setup of the Nakajima test is symmetric. Assuming that the failure zone evolves symmetrically, i.e., along the vertical center line, it is sufficient to only simulate a quarter of the full geometry and to rebuild the full solution by mirroring; see Fig. 7 (left). This is only exact, if the RVEs are also symmetric, since mirroring of the macroscopic solution also implies mirroring of the RVEs; see Fig. 7 (right). Hence, for an asymmetric RVE, we violate the assumption of a periodic unit cell because mirroring leads to a change in orientation for all four quadrants. In this case, the solution generated by the symmetric assumption is only an approximation to the solution of the simulation using the full geometry of the sample sheet, even for the first quadrant of the full geometry. Nevertheless, we use the symmetric assumption throughout this article, even when the RVEs are asymmetric, to reduce the number of MPI ranks by 75%; see Sect. 3.4. As a sanity check, we have also performed simulations for the full geometry; these will, however, be presented elsewhere due to space limitations.

For the simulation of one quarter, we have to add further boundary conditions along the symmetric boundaries of the quarter. Then, we have to fix all y-coordinates of macroscopic material points $\overline{p}$ with $\overline{p}_y = \overline{c}_y$. Analogously, we have to fix all x-coordinates for macroscopic points with $\overline{p}_x = \overline{c}_x$; see Fig. 7 (left) and Sect. 4.1.

4.3 Failure Criterion

For the detection of macroscopic crack initialization, we have to formulate a failure criterion and a maximum critical value, which will indicate the initialization of failure. Note that, for the computation of the forming limit, we do not need to simulate cracks or crack propagation. Instead, we are only interested to compute when structural failure occurs.

We use the Cockcroft and Latham criterion [19],

$$\overline{W}_k = \overline{W}(\overline{\alpha}_k) = \int_0^{\overline{\alpha}_k} \max\left(\overline{\sigma}_I(\overline{\alpha}), 0\right)\, d\overline{\alpha}, \tag{3}$$

which was used by Tarigopula et al. [93] for analyzing large deformation in DP steels. It depends on the maximum principal stress component $\overline{\sigma}_I$ and the equivalent plastic strain $\overline{\alpha}_k := \overline{\alpha}(t_k)$ at load step k (pseudo time t_k) and is integrated over $\overline{\alpha}$. Since $\overline{\alpha}$ depends on the load step, this also holds for the failure criterion $\overline{W}$ and the stress $\overline{\sigma}$. In general, in FE^2, $\overline{\alpha}_k$ is not known explicitly but can be approximated by the volumetric average[4] $\widetilde{\alpha}_k := \langle \alpha(t_k) \rangle$ from the microscopic level at load step k.

The value of $\overline{W}$ is computed at each Gauß point and is accumulated throughout the loading process until at least one Gauß point exceeds a critical value $\overline{W}_c$ at which failure initializes, i.e., $\overline{W} \geq \overline{W}_c$ is fulfilled. Tarigopula et al. provide the value $\overline{W}_c = 590 - 610$ MPa for DP800 grade of steel; see [93]. Since we consider DP600 grade of steel, the critical value should be smaller in our case.

Equation (3) is approximated by numerical integration and using $\widetilde{\alpha}$

$$\begin{aligned}\overline{W}_k \approx \overline{W}(\widetilde{\alpha}_k) &= \int_0^{\widetilde{\alpha}_k} \max\left(\overline{\sigma}_I(\widetilde{\alpha}), 0\right)\, d\widetilde{\alpha} \approx \sum_{i=1}^{k} \max\left(\overline{\sigma}_I(\widetilde{\alpha}_i), 0\right) \cdot (\widetilde{\alpha}_i - \widetilde{\alpha}_{i-1}) \\ &= \overline{W}_{k-1} + \max\left(\overline{\sigma}_I(\widetilde{\alpha}_k), 0\right) \cdot (\widetilde{\alpha}_k - \widetilde{\alpha}_{k-1}).\end{aligned} \tag{4}$$

where $(\widetilde{\alpha}_i - \widetilde{\alpha}_{i-1})$ is nothing else than the increment of the equivalent plastic strain from load step $i-1$ to load step i and $\overline{W}_0 = \widetilde{\alpha}_0 = 0$. Hence, we can sum up the failure criterion $\overline{W}$ over all load steps and summation is performed whenever a load step reached convergence. See Fig. 9 for an example of the evolution of the failure criterion $\overline{W}$ during a Nakajima simulation.

Let us note that the failure criterion is formulated such that ductile failure takes place due to tensile stresses and shear stresses, where the effect of tensile stresses is considered by usage of the maximum positive principal value of Cauchy stress and the effect of shear stresses and work-hardening is considered through the equivalent plastic strain increment.

4.4 Numerical Realization of the Experimental Cross Section Method

In the experiment, major and minor strains $\overline{\varepsilon}_1$ and $\overline{\varepsilon}_2$ are evaluated at the top surface of the sample sheets along cross sections. However, the simulation only provides

[4]Note that here and in the following, all volumetric averages obtained from microscopic level are in brackets $\langle\ \rangle$.

www.dbooks.org

exact macroscopic values in the integration points, which generally do not coincide with the finite element nodes. Therefore, the simulations do not provide any exact macroscopic values on the sample sheets surface, and we decided to consider cross sections along those Gauß points closest to the upper surface.

For the computation of major and minor strains $\overline{\varepsilon}_1$ and $\overline{\varepsilon}_2$ we transform our resulting 3D strain tensor to the 2D plane strain tensor and then follow the strategy as described in [97]. For further details, we refer to [95].

In this article, we show results for computations using the symmetric test setup of the experiment; see Sect. 4.2. This automatically implies that we assume that the failure zone evolves along the vertical center line and the center of the failure zone is identical to the center of the upper surface of the sample. For the computations using symmetry, all discretizations have finite element nodes at the center of the probe. Keeping in mind that we choose cross sections along Gauß points, no cross section cuts through the center of the failure zone. To keep the distance between the first cross section and the center of the failure zone as small as possible, we consider integration points with the smallest distance to the horizontal center line as first cross section. For simplicity, the other cross sections are along the remaining Gauß points of the same finite elements which were used for the first cross section. Hence, the distance between the cross sections depends on the diameter of those finite elements and is smaller than 2 mm in our computations.

Due to the symmetric computations, we only have one side of the cross sections at hand but the other side can be simply generated by mirroring; see Fig. 2 and the upper pictures in Fig. 8.

Note that the cross section method can only be used for specimens with a single failure zone. Unfortunately, in our simulations the failure zone does not always evolve along the vertical center line but parallel to it for sample sheets with a parallel shaft width of 100 mm or more. Hence, mirroring leads to the occurrence of a second failure zone; see Figs. 11 (left) and 12. For these specimens, first simulations using the upper half or the complete domain of the sample sheet also lead to two failure zones parallel to the vertical center line. For simulations obtaining two failure zones, we assume that the maximum major strain along the cross section defines the position of the failure zone. Neglecting all values between the vertical center line and the maximum major strain and shifting the failure zone to the vertical center line, we can proceed as before; see Fig. 8.

So far it is not clear, whether the symmetry assumptions hold for specimens with failure zones parallel to the vertical center line. In future work, we have to perform a comparison with the full geometry.

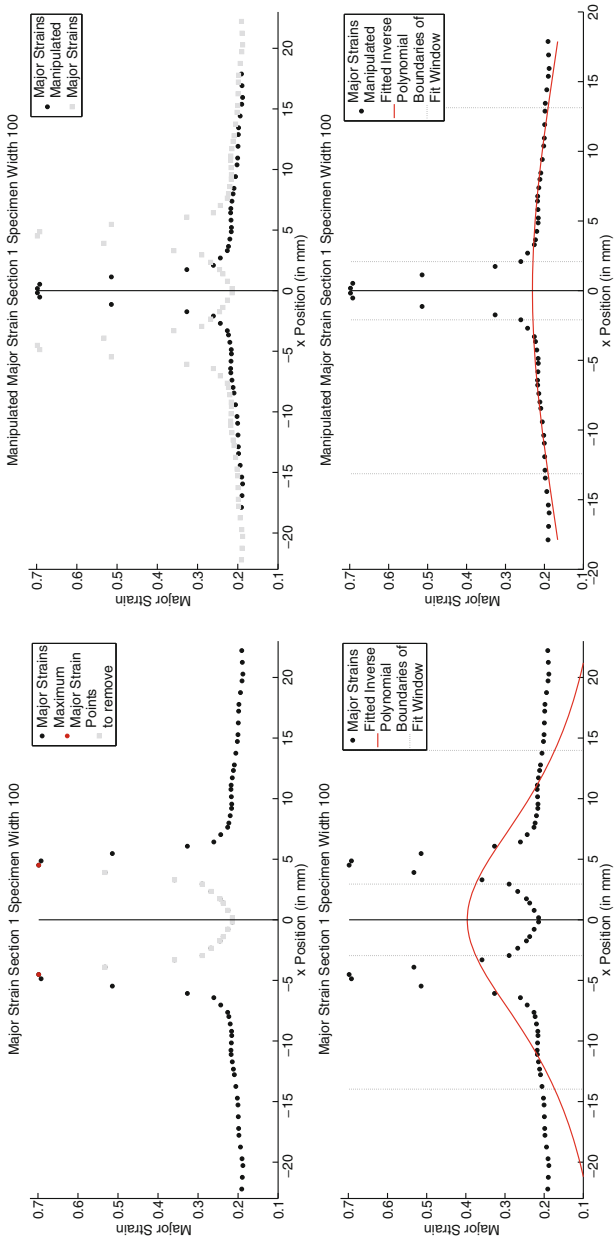


Fig. 8 Major strains along the first cross section for the specimen with a width of the parallel shaft of 100 mm. See also the description of the cross section method in Sect. 2. **Top:** Original values (left) and major strain values after shifting maximum values to the center as described at the end of Sect. 4.4 (right). **Bottom:** Fitted inverse second-order polynomials of original (left) and shifted (right) major strains. Computation of optimal fit windows is described in [77, Ch. 5.2.3, 5.2.4]

www.dbooks.org

5 The Virtual Forming Limit Diagram Computed with FE2TI

Since we use the symmetry assumptions for the numerical simulation of an FLC, we perform our simulations on a quarter of the full specimen and rebuild the solution by mirroring; see also Sect. 4.2. The computational domain is discretized by structured Q2 finite elements, where the number of finite elements depends on the width of the parallel shaft as well as on the chosen boundary conditions; see Sect. 4.1 as well as Table 2. Note that we use (only) two Q2 finite elements in the direction of thickness for all specimens.

For the microscopic problems, we consider an SSRVE with two ellipsoidal inclusions discretized by 1470 P2 finite elements and 7152 d.o.f. in an unstructured manner. As mentioned before, the resulting tangent problems on the microscopic level are solved by using the direct solver package PARDISO [81] and each problem is solved on an individual MPI rank. Hence, the number of macroscopic finite elements automatically determines the number of required MPI ranks, which is 27 times the number of finite elements. We have performed all our simulations on JUWELS [45] using 2 MPI ranks per core and a penalty parameter of 500. For the specifications of the rigid tools, we refer to Sect. 4.1.

As an initial load increment, we choose 0.1 mm and define $\overline{l}^{\max} = 0.2$ mm as maximum allowed load increment. Our stopping criterion is either based on reaching a predefined covered distance of the forming tool of 40 mm or on the load increment and not on the failure criterion, since we have only little experience how to choose the critical value $\overline{W}_c$ to detect failure. Simulation stops if the load increment of 10 successive load steps is smaller than a predefined allowed minimum, which is the initial load increment multiplied by 10^{-4}, or, if the load increment has to be reduced 7 times within a single load step. This is motivated by the fact that small load increments indicate hard numerical problems which are expected in case of material failure.

We have summarized some data on our Nakajima simulations including the number of restarts in Table 2. Note that most restarts are caused by reaching the requested wall time and only in few cases, if any, by hardware errors.

For all specimens with a parallel shaft, we obtain comparable results, which are characterized in the following. After a certain covered distance of the tool, a local increase in the failure value $\overline{W}$, the major strains $\overline{\varepsilon}_1$, the equivalent plastic strain $\widetilde{\alpha}$, the thinning rate, and the von Mises stress can be detected almost simultaneously along an area parallel to the vertical center line. Later, the values continue to rise, especially in this area, so that the degree of localization increases; see Fig. 9. Finally, some microscopic problems within the aforementioned localized area cause the simulation to end. At this point, however, a pronounced change in thickness can be observed within the localized area, which can be associated with material failure; see cross sections in Figs. 12 and 10 (top right) as well as the upper right picture in Fig. 11 (left).

Table 2 Nakajima simulations with different specimens

	Width 30	Width 40	Width 50	Width 70	Width 90	Width 100	Width 110	Width 115	Width 129	Full circular
Macro finite elements (Q_2)	424	392	460	482	558	558	580	574	574	542
MPI ranks	11,448	10,584	12,420	13,014	15,066	15,066	15,660	15,498	15,498	14,634
Macro d.o.f.	13,725	12,705	14,804	15,465	17,835	17,835	18,495	18,195	18,195	17,145
Covered dist. punch (mm)	27.156	28.325	29.242	29.896	30.734	31.654	32.593	35.502	36.566	40.000
Load steps	736	751	806	901	985	898	780	1358	1651	569
Newton its.	8148	8510	9272	9823	10,976	8604	8540	9556	10,012	8064
Runtime (h)	17.60	16.00	19.35	22.00	27.00	24.00	21.00	25.25	24.17	19.25
Restarts	1	0	1	1	2	1	1	2	1	1
Overhead load steps	74	–	6	8	68	16	24	104	3	12
Overhead runtime (h)	1.70	–	0.23	0.23	1.71	0.40	0.58	2.46	0.25	0.41

Microscopic problems: SSRVE with two ellipsoidal inclusions; 1470 unstructured P2 finite elements and 7152 d.o.f. Two MPI ranks per core; computed on JUWELS [45]; one microscopic problem computed per MPI rank. Overall problem size is obtained by multiplying number of MPI ranks by d.o.f. of microscopic SSRVE and adding number of macroscopic d.o.f. resulting in 80–112 million d.o.f

www.dbooks.org

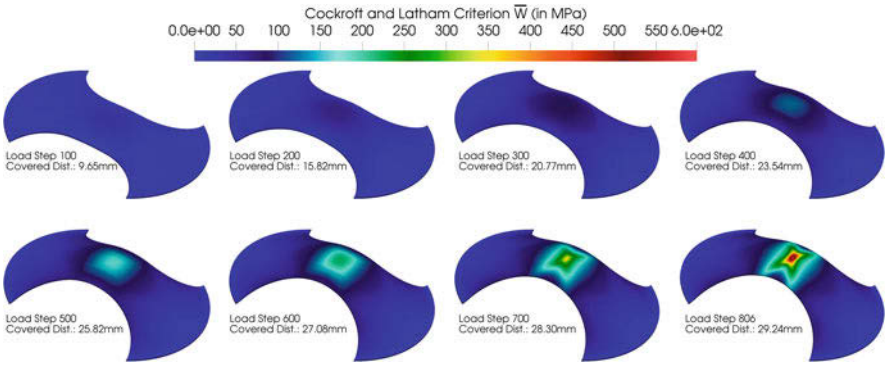


Fig. 9 Evolution of failure criterion $\overline{W}$ during the simulation for the specimen with a width of the parallel shaft of 50 mm

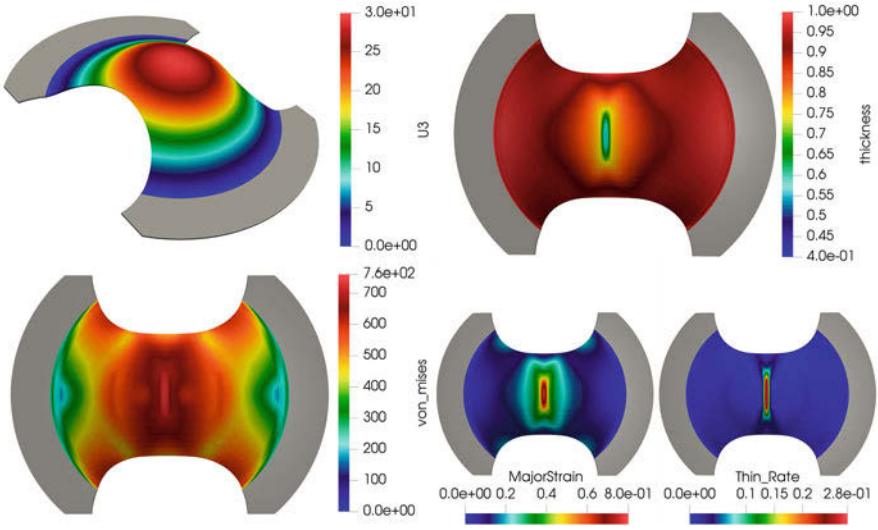


Fig. 10 Final solution of the simulation with a specimen with a width of the parallel shaft of 70 mm; z-direction (top left), thickness (top right), von Mises stress (bottom left), major strain (bottom center), and thinning rate (bottom right). The grey area is not simulated since material is totally clamped between die and blank holder

For the completely circular specimen we do not observe any localized effects along an area parallel to the vertical center line, even if we reach the predefined distance of 40 mm. Instead, we obtain a similar behavior over nearly the complete contact area; see Fig. 11 (right).

The results for specimens with a parallel shaft can be divided into two groups. The first group contains all samples with a parallel shaft width of at most 90 mm. For these specimens material failure occurs along the vertical center line; see Fig. 10.

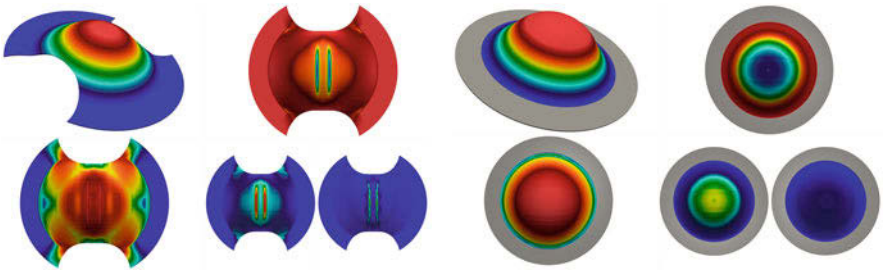


Fig. 11 Final solution of the simulation with a specimen with a width of the parallel shaft of 100 mm (**left**) and the completely circular specimen (**right**); variables and color bars as in Fig. 10. **Left:** Material between blank holder and die is simulated since material movement is allowed. **Right:** Material between blank holder and die is assumed to be clamped (dark grey) and hence is not simulated

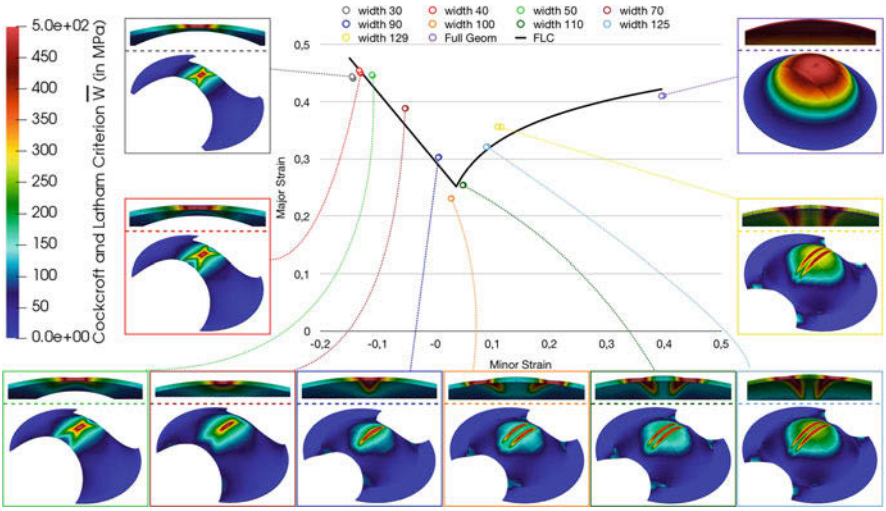


Fig. 12 Final results of distribution of Cockcroft and Latham failure value $\overline{W}$ for all Nakajima simulations and associated Forming Limit Diagram (FLD) with FLC (black curve) for $\overline{W}_c = 450$. In the cross section, one can identify local necking in thickness for all but the completely circular specimen

All specimens with a wider parallel shaft belong to the second group, which can be characterized by material failure parallel to the vertical center line. For these samples, mirroring of the computed solution leads to the occurrence of a second failure zone; see Figs. 11 (left) and 12. As mentioned before, first numerical tests indicate that this also holds when simulating the complete specimen or the upper half of it. Hence, we decided to use the results to determine an FLC. The adaptions to evaluate major and minor strains for failure zones parallel to the vertical center line are mentioned in Sect. 4.4; see also Fig. 8. Without manipulating the values along the cross sections, we would obtain unphysically high values in the FLD.

www.dbooks.org

Based on all available results we have subsequently defined a critical value of $\overline{W}_c = 450$ for the failure criterion. Hence, we have to find for all simulations the corresponding load step for which the failure value in at least one point exceeds the critical value $\overline{W}_c = 450$ for the first time; see Table 3.

When we have found the corresponding load steps, we have to compute the major and minor strain values along the cross sections perpendicular to the failure zone and generate for each cross section one point in the FLD. So, we come up with 30 different pairs of major and minor strains wherein each three belong to one specimen; see Fig. 12. The final FLC is derived by regression; see black curve in Fig. 12 and [95] for details.

6 Linear and Nonlinear Parallel Solvers

For large scale computations using FE2TI, scalable parallel implicit solver algorithms are needed for the problems on the microscale as well as the macroscale [64]. Another focus of the EXASTEEL project therefore was on solver algorithms, i.e., efficient and highly parallel scalable implicit solvers for linear and nonlinear problems arising from a finite element discretization of linear and nonlinear partial differential equations; see, e.g., [5, 54–57, 59–61, 63].

Here, nonlinear domain decomposition (DD) approaches, namely nonlinear FETI-DP (Finite Element Tearing and Interconnecting-Dual Primal) and nonlinear BDDC (Balancing Domain Decomposition by Constraints) methods, have been introduced in the first phase of EXASTEEL and improved during the second phase, where also new variants were introduced.

In [64], our new nonlinear FETI-DP solver algorithms were then applied within large FE2TI simulations for the first time: We have used Nonlinear-FETI-DP-1 as a parallel implicit solver for the microscopic problems using 114,688 KNL cores of the Theta many-core supercomputer (Argonne National Laboratory) [64, section 4.9]. However, the nonlinear DD methods developed in the project have a broad range of applications, e.g., in hyperelasticity, elasto-plasticity, or nonlinear diffusion problems.

To further improve the performance of the nonlinear solvers, also the efficiency and parallel scalability of all building blocks was in the focus of the EXASTEEL project [5, 54–56, 60, 61].

In this section, we describe very briefly our nonlinear domain decomposition approaches and sum up the achievements and highlights obtained within the past 6 years.

Table 3 Identification of first load step with failure values higher than the critical value $\overline{W}_c = 450$; two MPI ranks per core; computed on JUWELS[45]

	Width 30	Width 40	Width 50	Width 70	Width 90	Width 100	Width 110	Width 115	Width 129	Full circular
Covered dist. punch (mm)	26.725	27.813	28.747	29.689	30.361	30.986	31.972	34.976	36.332	38.488
Load step	684	710	743	873	896	785	701	768	800	541
Newton its.	7708	8099	8737	9695	10,635	8318	8253	8931	9142	7676
Maximum $\overline{W}$	454.49	451.15	451.33	451.24	450.24	450.08	451.92	451.27	450.14	450.91
Runtime (h)	16.55	15.16	17.96	21.39	26.14	23.09	19.80	23.09	22.36	18.15

www.dbooks.org

6.1 Nonlinear FETI-DP Framework

Classical domain decomposition methods are robust and highly scalable divide-and-conquer algorithms for the iterative solution of discretized linear partial differential equations. In the case of FETI-DP methods [27, 28, 47–50], the computational domain is decomposed into nonoverlapping subdomains which are distributed to the available compute cores. FETI-DP methods are well established in structural mechanics and have been awarded a Gordon Bell price [10].

The robustness and scalability originates from the sparse direct solvers applied on the subdomains, combined with a carefully designed coarse problem. The coarse problem, though necessary for robustness, is a potential scaling bottleneck, since its size grows with the number of subdomains, i.e., with the number of parallel cores. In order to retain scalability, the coarse solution can be approximated, e.g., by algebraic multigrid methods; see [46, 48]. Finally, to solve nonlinear problems, a linearization with Newton's method is usually applied first, and the linearized tangential systems are then solved, e.g., by FETI-DP. We refer to the latter approach as Newton-Krylov-FETI-DP.

In contrast, in nonlinear FETI-DP or BDDC methods, first introduced in [51], the discretized nonlinear partial differential equation is decomposed into small and independent nonlinear problems before linearization. In the case of nonlinear FETI-DP, a nonlinear coarse problem is added by strongly coupling the local nonlinear problems in few variables on the interface of the domain decomposition, as, e.g., vertices or averages over edges or faces. This leads to a nonlinear FETI-DP saddle point system; see, e.g., [51, eq. (3.2)], [54, eq. (3.4)], or [59, eq. (1)]. Also, a partially nonlinear elimination process of sets of variables from the nonlinear FETI-DP saddle point system is possible before linearization. The nonlinear elimination process can be interpreted as a nonlinear right-preconditioner, which we described in [59, Section 2.5] in detail. There we also introduced a unified framework to describe different nonlinear FETI-DP and BDDC methods. The selection of the elimination set finally defines the nonlinear FETI-DP method precisely. We discussed four canonical choices in [59], but other choices are feasible and possibly beneficial. Let us briefly repeat the four variants from [59]. In FETI-DP, all degrees of freedom or variables are divided into three classes. First, all variables belonging to the interior of the subdomains are grouped into the set I (marked as circles in Fig. 13), second, all variables of the global coarse problem are grouped into the set Π of so-called primal variables (marked as squares in Fig. 13), and third, all local degrees of freedom on the interface are grouped into the set Δ of so-called dual variables (marked as dots in Fig. 13). Let us remark that continuity of the solution in the dual degrees of freedom is enforced iteratively by enforcing zero jump constraints with a Lagrangian approach; see [27] for details. Finally, Nonlinear-FETI-DP-1 is defined by eliminating nothing, Nonlinear-FETI-DP-2 by eliminating everything, Nonlinear-FETI-DP-3 by eliminating the interior and dual variables, and finally Nonlinear-FETI-DP-4 by eliminating the interior variables; see Fig. 13 for a visualization of the different variants.

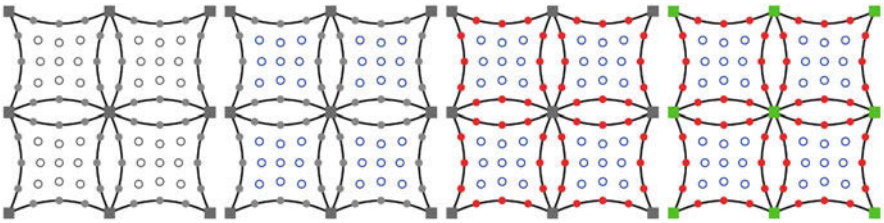


Fig. 13 The coarse problem is marked with squares, the interior degrees of freedom by circles, and the dual degrees of freedom by dots. The colored degrees of freedom are eliminated nonlinearly before linearization in the different variants. **From left to right:** Nonlinear-FETI-DP-1, Nonlinear-FETI-DP-4, Nonlinear-FETI-DP-3, Nonlinear-FETI-DP-2

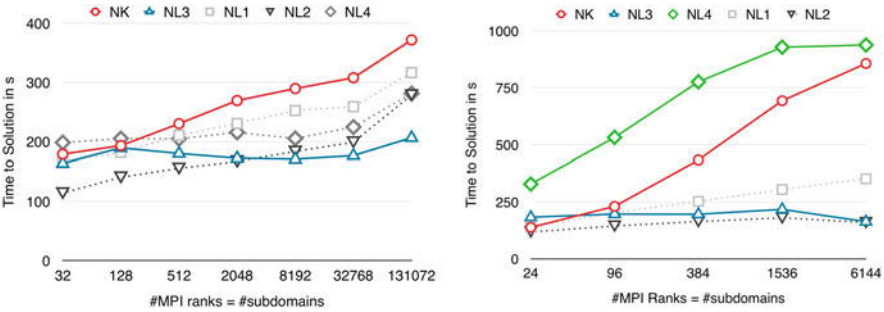


Fig. 14 Comparison of classical Newton-Krylov-FETI-DP (NK), Nonlinear-FETI-DP-1 (NL1), Nonlinear-FETI-DP-2 (NL2), Nonlinear-FETI-DP-3 (NL3), and Nonlinear-FETI-DP-4 (NL4) using the JUQUEEN supercomputer. **Left:** Nonlinear p-Laplace inclusions not touching the interface; 40k d.o.f. per subdomain; **Right:** Nonlinear p-Laplace channels crossing the interface; 160k d.o.f. per subdomain. For a detailed discussion of the results see also [59, Fig. 10 and 12] and corresponding descriptions. If an appropriate nonlinear elimination set is chosen for the problem (as is the case here in NL3) then the nonlinear method outperforms the classical Newton-Krylov approach significantly

If the elimination set is chosen appropriately, nonlinear FETI-DP methods can outperform classical methods, i.e., Newton-Krylov-FETI-DP. In [42], a heuristic approach is suggested to eliminate the areas with strong nonlinear effects. For illustration let us consider combinations of the nonlinear p-Laplace equation and the linear Laplace equation, where the nonlinearities either touch the subdomain interface or are at a distance. In the latter case, we choose nonlinear inclusions of p-Laplace enclosed in the subdomains and in the first case nonlinear channels of p-Laplace cutting certain edges. For a detailed description of the chosen model problem, see [59, Section 5.1, Fig. 7]. Considering the inclusions, all nonlinear FETI-DP methods work well. Considering the example with channels, it is necessary to at least eliminate the dual interface variables; see Fig. 14 or [59]. Let us note that Nonlinear-FETI-DP-1 performs well in both cases, which is a result of a careful choice of the initial value; see [51, Section 3.3] for details.

www.dbooks.org

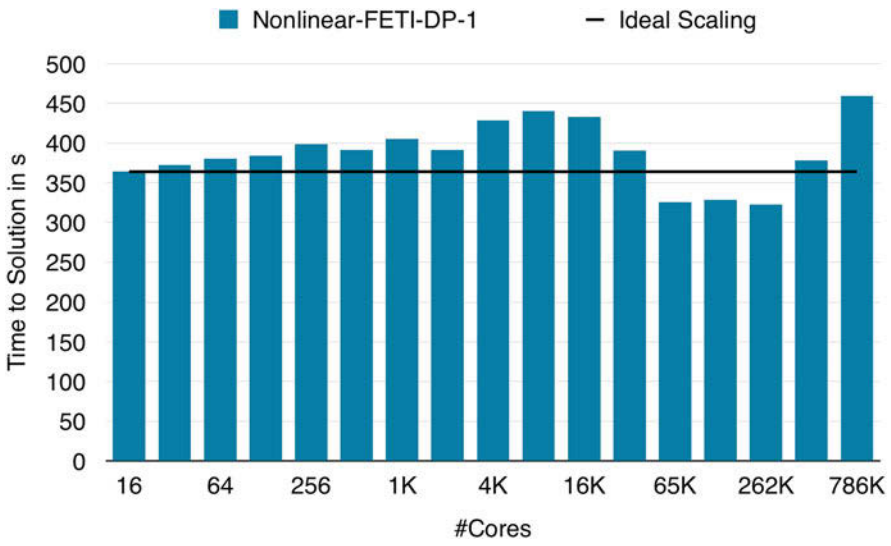


Fig. 15 Weak scalability of Nonlinear-FETI-DP-1 for a p-Laplace problem computed on the Mira supercomputer; largest problem has 62.9 billion d.o.f.; results from [57]; see also [57, Fig. 4 (left)]

Another benefit of nonlinear domain decomposition approaches is the localization of work, which increases the scalability of these methods. This can be verified either in Fig. 14 or in our larger weak scaling experiments on Mira published in [57] and presented in Fig. 15, where an algebraic multigrid preconditioner from the BoomerAMG package [38] is used to approximate the coarse solve to obtain scalability without losing robustness.

We have also considered approaches to improve the convergence of nonlinear FETI-DP methods. We have introduced heuristics to determine if a nonlinear elimination is useful in a certain Newton step or not. Additionally, the elimination process is approximated up to a necessary tolerance to save computational cost. This approach is called NL-ane (approximate nonlinear elimination) and is also discussed in [59, 62]. We recently also considered a globalization strategy using the SQP approach; see Sect. 6.1.2.

Finally, we successfully investigated a hybrid parallelization of FETI-DP using PARDISO in [55], and also considered nonlinear FETI-DP and BDDC methods where the sparse direct solvers are replaced by preconditioners; see [56, 60, 61].

6.1.1 Improving Energy Efficiency

In classical Newton-Krylov methods, global synchronization occurs in every Krylov iteration. Global synchronization can be significantly more coarse-grained in our nonlinear domain decomposition methods since the Krylov iteration can be asyn-

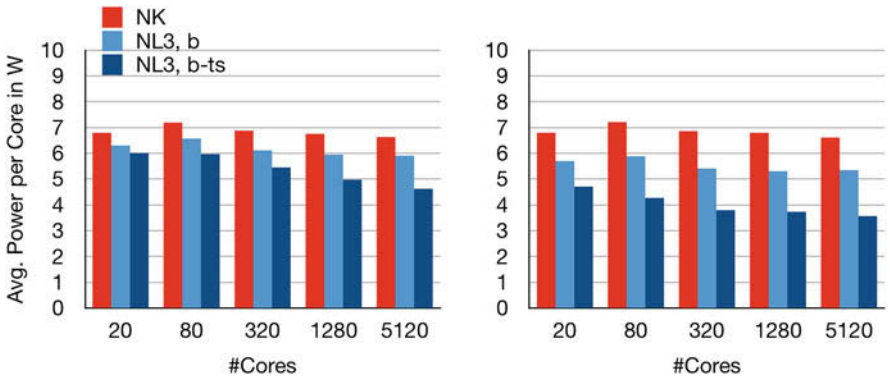


Fig. 16 Power per core for Newton-Krylov-FETI-DP (NK) and Nonlinear-FETI-DP-3 (NL3) with normal barrier (b) and test-sleep approach (b-ts) on Meggie cluster. Each subdomain problem has 160k d.o.f. **Left:** Nonlinearity in each of the subdomains; **Right:** Nonlinearity in a single subdomain. See also [63, Fig. 10] and corresponding descriptions for details

chronous between subdomains. In this section, we describe how this can be exploited to save energy when load imbalances occur.

If the nonlinear elimination set in nonlinear FETI-DP is completely local to the subdomains as, e.g., in Nonlinear-FETI-DP-3, the nonlinear subdomain problems can be solved in parallel and asynchronously. This solution process using Newton's method can introduce a load imbalance, even for perfectly balanced meshes, if the number of Newton iterations is different between subdomains; see [63, Fig. 7]. Note that even for these cases, Nonlinear-FETI-DP-3 typically has a shorter time to solution compared with classical approaches; see Fig. 14.

In [63], we have suggested to use a nonblocking barrier in combination with a sleep statement to set idling cores in deep sleep states, to reduce energy consumption. This is feasible in nonlinear parallel domain decomposition since the synchronization between the cores is coarse grained (typically larger than 1 s). During these phases sleeping cores wake up every 10 ms. The wake-up latency itself for current Intel processors is significantly below 1 ms. Therefore, the overhead of the sleep and wake up approach is insignificant compared to the time spawn of global synchronization and does not impact overall performance or scalability. We call this method test-sleep approach. To investigate the energy saving potential, we measured the energy consumption of Nonlinear-FETI-DP-3 in [63] reading out the RAPL hardware counters with LIKWID [94] on the Meggie[5] cluster at Erlangen Regional Computing Center (RRZE). In Fig. 16, we present the power consumption per core for two different scenarios, i.e., a single nonlinear inclusion in a single subdomain or a single nonlinear inclusion in each subdomain. The total energy

[5]Standard Cluster with Intel Omnipath Interconnect and two Intel Xeon E5-2630 v4 chips per node.

www.dbooks.org

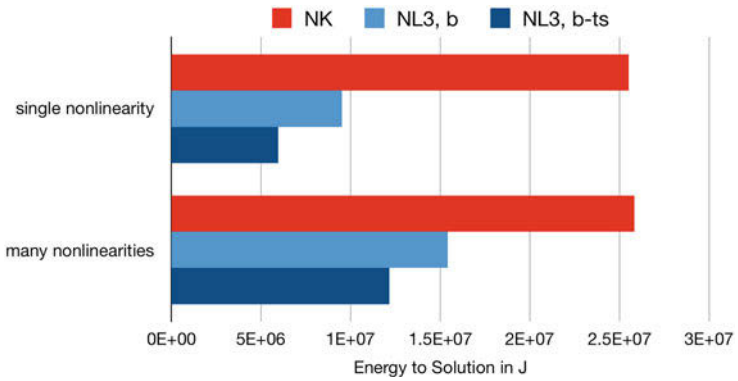


Fig. 17 Total energy to solution for Newton-Krylov-FETI-DP (NK) and Nonlinear-FETI-DP-3 (NL3) with normal barrier (b) and test-sleep approach (b-ts). Computation on 5120 Meggie cores. See also [63, Fig. 9] for a complete weak scaling study

consumed by the nodes during the solution procedure on 5120 cores is shown in Fig. 17. The test-sleep approach also works for alternative nonlinear domain decomposition methods, as, e.g., ASPIN [17]; see [63] for a brief discussion using the ASPIN implementation described in [15] which is provided in PETSc.

6.1.2 Globalization

We consider different globalization strategies for our nonlinear domain decomposition methods. For the different nonlinear FETI-DP methods, we consider trust region methods and also an approach based on the SQP (sequential quadratic programming) method using the exact penalty function $P_\beta^1(\tilde{u}) = J(\tilde{u}) + \beta||B\tilde{u}||_1$, where J denotes the mechanical energy and $B\tilde{u}$ are the FETI-DP equality constraints; see Table 4 for some results.

Table 4 Number of global iterations for a snap through buckling problem for compressible Neo-Hookean energy with material parameters $E = 210$, $\nu = 0.3$ in 2D; 100 subdomains, 20,402 degrees of freedom; $-$: failed

	SQP		Trust region		Without globalization	
Volume force	NL-1	NL-4	NL-1	NL-4	NL-1	NL-4
$\left[0, -20\right]^T$	6	6	321	343	–	–
$\left[0, -40\right]^T$	6	6	298	469	–	–
$\left[0, -80\right]^T$	7	7	258	662	–	–

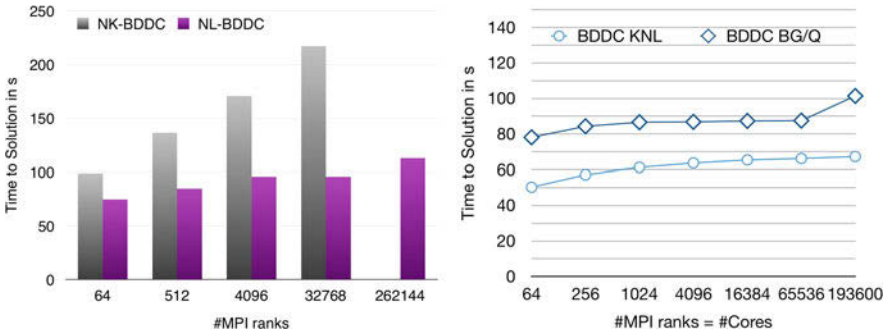


Fig. 18 Left: Weak scaling experiment on the JUQUEEN supercomputer; deformation of a hyperelastic cube with 0.7 billion d.o.f. for the largest computation (heterogeneous Neo-Hooke material model); using Newton-Krylov-BDDC or Nonlinear-BDDC; see [60, Section 4.4, Fig. 15] for details. For 262,144 subdomains Newton-Krylov-BDDC did not converge within 20 Newton iterations. **Right:** Weak scaling of linear BDDC solver with approximate coarse solve (using AMG) on JUQUEEN (BG/Q) and Theta (KNL) supercomputers for a heterogeneous linear elastic model problem in two dimensions with 14k d.o.f. per subdomain; see [60, Fig. 7] for details

6.2 Nonlinear BDDC Methods

Using the same elimination set as in Nonlinear-FETI-DP-4, the nonlinear BDDC method [51] can be derived, which is based on its linear version; see [20, 24, 71–73]. We presented an efficient and scalable implementation of linear and nonlinear BDDC avoiding the computation of Schur-complements in [60]. This approach proved to be faster, more scalable, and more robust for nonlinear hyperelasticity problems (see Fig. 18 (left)) as well as for elasto-plasticity problems using realistic RVE microstructures obtained from EBSD measurements; see [60, Table 4.7]. We also studied the scalability of the embedded linear BDDC solver on different architectures; see Fig. 18 (right).

7 Multicore Performance Engineering of Sparse Triangular Solves in PARDISO Using the Roofline Performance Model

The PARDISO [12, 23, 69, 96] parallel sparse direct solver is a building block in FE2TI. As long as the macroscopic problem is small enough, it can be solved directly by PARDISO; if the microscopic problems are of reasonable size, it is efficient to use PARDISO concurrently on the microscale problems. For large micro and macro problems, the direct solver has to be replaced by linear or nonlinear FETI-DP or BDDC domain decomposition solvers. Here, PARDISO is typically applied as the subdomain and coarse solver.

www.dbooks.org

The PARDISO solver has two phases: factorization and forward/backward substitution, with factorization being more time consuming than substitution. However, the former is only performed once in a FETI-DP or BDDC domain decomposition iterative process, whereas the latter is repeated many times. We are in particular interested in the forward and backward solution process of sparse direct solvers since they build the computational kernel, e.g., in FETI-DP or BDDC methods. FETI-DP and BDDC are known to be highly parallelizable, but most implementations are using sparse direct solvers as building blocks. More precisely, most domain decomposition implementations use sparse direct solves for the local subdomain problems to obtain the necessary robustness. Additionally, the coarse problem is usually solved directly up to a certain size, but for larger problems the coarse solve is often approximated by, e.g., AMG or recursive applications of the domain decomposition approach itself. Therefore, we investigate and analyze the performance of the forward/backward solution process of PARDISO for the local subdomain solves in FETI-DP and BDDC and present not only numerical results, but also a detailed performance analysis for a representative sparse solver kernel based on the roofline model. The goal is to create an analysis of this part of the algorithm and to establish a roofline performance model [98], which considers performance bounds given by the memory bandwidth and the processor peak performance. We modeled both the serial and parallel execution phases. Despite the fact that the roofline model prediction can be inaccurate in the serial case, when the in-core execution or in-cache transfers become dominant, it still provides an easily obtainable upper bound. The simple roofline model brings together the computational capabilities of the processor and the available memory bandwidth with the requirements of an algorithm. In our case the relevant quantities are the number of FLOPs performed and the data transferred between processor and memory, which we determined by an analysis of the forward/backward substitution.

The performance of the forward and backward substitution process is analyzed and benchmarked for a representative set of sparse matrices on modern x86-type multicore architectures. The characteristic quantities of these systems are shown in Table 5 along with the required machine specific input parameters (lower part of Table 5) for the roofline model. The measurement approach, its validation, as well as limitations are discussed in [98]. Our modeling approach covered both the serial and parallel execution phases, allowing for in-socket performance predictions. The performance modeling results for a discretized Laplacian matrix ('lapl2') and a local subdomain matrix ('BDDC') from the EXASTEEL project are shown in Fig. 19; see also Table 6 for dimensions and numbers of nonzeros for the considered matrices. The latter matrix is used in FETI-DP as well as BDDC methods to compute the discrete harmonic extension from the domain decomposition interface to the interior of a certain subdomain. The specific problem represents a typical RVE using the J_2 elasto-plasticity model including the material parameters also used in the computations of the FLD. We verified that the considered subdomain already showed a plastic behavior.

As opposed to the original roofline model, the modified roofline model covers also the performance impact of limited scalability imposed by the algorithm, i.e.,

Table 5 Details of the Intel and AMD hardware systems evaluated

Name	IVB	BDW	SKX	ZEN-S
Processor name	Intel Xeon	Intel Xeon	Intel Xeon	AMD EPYC
	E5-2660 v2	E5-2630 v4	Gold 6148	745
Micro architecture	Ivy Bridge	Broadwell	Skylake	Zen
Frequency [GHz]	2.2	2.2	2.4	2.3
Number of cores	10	10	20	24
Vector instruction set	AVX	AVX2	AVX-512	AVX2
NUMA LDs	1	1	1	4
Scalar read bandwidth				
1 core [GB/s]	9.5	11.5	14.5	19.3
NUMA LD [GB/s]	44.4	56.3	108.0	37.6
Scalar ADD+MUL/FMA				
1 core [F/cy]	2	4	4	4
NUMA LD [F/cy]	20	40	80	24
Scalar machine balance B_m				
1 core [B/F]	2.2	1.3	1.5	2.1
NUMA LD [B/F]	1.0	0.6	0.6	0.7

The NUMA Locality Domain (LD) refers to a number of cores sharing a physical or logical memory controller

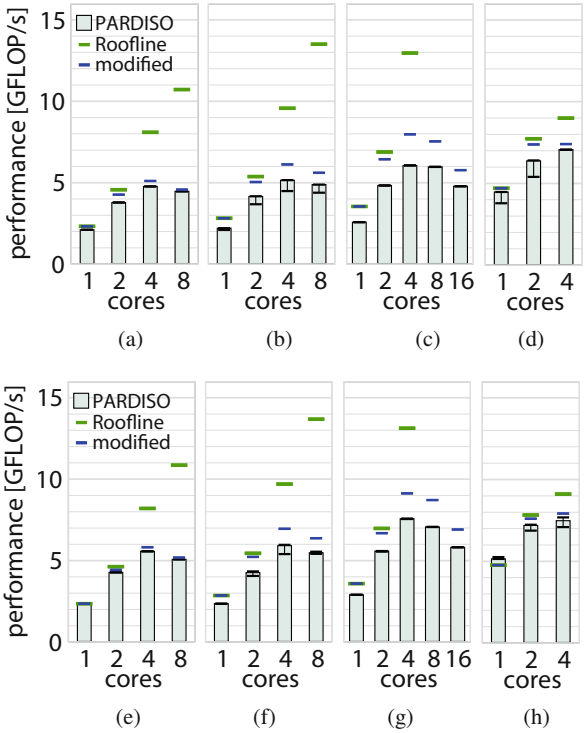


Fig. 19 Performance on IVB, BDW, SKX, and ZEN-S (left to right) for two matrices (upper and lower panel) from EXASTEEL. (**a**–**d**) lapl2. (**e**–**h**) BDDC. Copyright 2019, IEEE, [98]

www.dbooks.org

Table 6 Dimension (n) and number of nonzeros (nnz) for A and L for benchmark matrices

Matrix	n	nnz(A)	nnz(L)
lapl2	343×10^3	1×10^6	166×10^6
BDDC	750×10^3	31×10^6	1590×10^6

both serial and parallel execution phases of the forward and backward substitution are considered in the model; see [98] for details. It captures the behavior of the measured performance quite well compared to the original roofline model.

For Intel Ivy Bridge systems the modified roofline model error is only up to 5%. Further details are given in [98]. During the second phase of EXASTEEL the close collaboration with ESSEX-II in the context of performance engineering was also extended to iterative solvers, leading to a new promising recursive algebraic coloring scheme [1]. The benefits of the recursive algebraic coloring were demonstrated by applying it to the kernel operation of a symmetric sparse matrix-vector multiplication (SymmSpMV) on various multicore architectures. The performance was compared against standard multicoloring and various other algebraic block multicoloring methods. The coloring implementation resulted in an average and maximum speedup of 1.4 and 2, respectively. Our entire experimental and performance analysis process was also backed by the roofline performance model, corroborating the optimality of the approach in terms of resource utilization of the SymmSpMV kernel on modern hardware; see the ESSEX-II report in this book for details.

8 Improvement of the Mechanical Model for Forming Simulations

In this section, we describe improvements to the modeling developed in the project. Not all of the techniques described here are currently used in our FE2TI production simulations, mainly to reduce computational cost.

As mentioned earlier, the favorable macroscopic properties are to a large extent due to the heterogeneous microstructural features of the DP steels. A sophisticated heat treatment process is used to produce a heterogeneous microstructure with ferritic matrix and martensitic inclusions. This process is also accompanied by several effects which then in conjunction with the difference in mechanical properties of ferrite (soft phase) and martensite (hard phase) generate advantageous macroscopic properties. In this project, an initial volumetric strains approach, c.f. [14], was developed to mimic the micromechanical features resulting from the phase transformations during the production process.

The high yield and work-hardening properties of DP steels, on the other hand, pose a problem in terms of forming complex geometries and designing the metal working tools. One of the major issues associated with the forming of DP steels, is the large springback observed at the end of the forming process, which results in

undesirable geometries of the formed parts. Here, simulating the forming process with an accurate material behavior can help to predict springback precisely and further save valuable resources while optimizing the tooling parameters for the process. The springback behavior is found to be very closely associated to the Bauschinger factor of the material. Therefore, within this project a multiscale modeling strategy to effectively model the DP steel response under cyclic loading was developed. In this context, an efficient neural network based algorithm is employed to identify the associated microstructural material parameters, leading to a reduction in the required computational resources. In order to obtain further understanding of the correlation of the model parameters on the macroscopic behavior of DP steels during cyclic loading, uncertainty quantification studies have been carried out using the developed mechanical models.

Due to their higher accuracy and physical interpretability, crystal plasticity formulations may be used at the RVE level to directly describe plasticity in a polycrystal such as multiphase steels. Due to the fact that such formulations are computationally highly expensive, they may be primarily applied to computationally identify macroscopic yield surfaces. FE^2 simulations of metal forming processes based on such formulations at the small scale will however hardly be feasible. Therefore, as part of this project one goal was to improve the quality of micromechanical models to be used efficiently in the context of FE^2. The associated micromechanical simulations are mainly making use of a classical finite J_2 elasto-plasticity material model, c.f. [79, 87, 89, 90], which is used to model the micro-constituents (ferrite and martensite) by defining the hardening law

$$\beta^{\text{iso}} = y_{\infty}^{\text{iso}} + (y_0^{\text{iso}} - y_{\infty}^{\text{iso}})\exp(-\eta^{\text{iso}}\alpha) + h^{\text{iso}}\alpha. \tag{5}$$

Herein, y_0^{iso} is the initial yield stress, y_{∞}^{iso} is the saturation yield stress, η^{iso} is the exponent, h^{iso} is the linear hardening at saturation yield stress and α is the equivalent plastic strain variable. The material parameters of the models are calibrated based on uniaxial tests performed on the pure individual constituents martensite and ferrite. As representative microstructure a so-called statistically similar RVE was identified; see [14]. More information on SSRVEs can be obtained in [7, 8], and [83]. Although the individual phases can be represented accurately and the microstructure is reflected with high accuracy, still the experimental stress-strain response cannot be obtained. The main reason is that distributed properties in the ferritic matrix phase, which however result from the production process, were not yet taken into account. In addition to that, when focusing on cyclic loading protocols, the macroscopic kinematic hardening of the real sheet metal cannot be represented. As a suitable quantitative measure for the kinematic hardening, the so-called Bauschinger factor can be computed as $\overline{f}_{\text{B}} = (|\overline{P}_{\text{I}}| - \overline{P}_{\text{II}})/|\overline{P}_{\text{I}}|$. Herein, $\overline{P}_{\text{I}} = \overline{P}_x(\Delta\overline{l}_x/\overline{l}_{x,0} = -0.05)$ and $\overline{P}_{\text{II}} = \overline{P}_x(\Delta\overline{l}_x/\overline{l}_{x,0}(\overline{P}_x = 0) + 0.002)$, where $\boldsymbol{P}$ is the 1st Piola-Kirchoff stress tensor. Although the Bauschinger factor of the FE^2 simulation with $\overline{f}_B^{\text{comp}} = 0.47$ is interestingly high considering that for the individual phases no kinematic hardening is taken into account up to here, it does not agree well with the experimental value

www.dbooks.org

$\overline{f}_B^{\text{exp}} = 0.66$. Therefore, in this project three major improvements were developed to enhance the model quality for FE^2 simulations of sheet metal forming problems: (i) A mixed isotropic-kinematic hardening model was implemented at the microscale for the ferrite phase, (ii) an initial volumetric strains approach was developed to model the locally distributed plastic properties in the ferrite phase, and (iii) an implicit fit procedure was constructed based on a neural network to identify the kinematic hardening parameters.

A mixed hardening model was implemented for the ferritic phase, which consists of an exponential isotropic hardening law, see Eq. (5) and a linear kinematic hardening law, c.f. [89]. The yield criterion and the evolution of the back stress $\boldsymbol{\xi}$ are then given by

$$\phi^{\text{mix}} = ||\text{dev}\boldsymbol{\sigma} - \boldsymbol{\xi}|| - \sqrt{\frac{2}{3}}\beta^{\text{iso}} \quad \text{and} \quad \dot{\boldsymbol{\xi}} = \frac{2}{3}\dot{\lambda} H^{\text{kin}} \frac{\text{dev}\boldsymbol{\sigma} - \boldsymbol{\xi}}{||\text{dev}\boldsymbol{\sigma} - \boldsymbol{\xi}||}. \tag{6}$$

Here, H^{kin} is an additional material parameter, signifying the linear kinematic hardening. Thus, the material parameters associated with the ferritic phase should be newly identified for the mixed hardening material model. An appropriate multiscale approach has been developed which is described in Sect. 8.2.

8.1 Initial Volumetric Strains (IVS) Approach

The IVS approach proposed in [14] allows the modeling of heterogeneous yield stress distribution in ferrite and results in a good agreement of the predicted stress values with the experiments. Here, the ferritic yield curve is locally modified using modification factor $\gamma(\boldsymbol{X}) \in [1, 1.6]$, quantified based on physical and experimental observations. As a result of this continuous procedure, where the microstructure is subjected to first IVS, followed by subsequent mechanical loading, for, e.g., uniaxial tension, not only the distributed properties are obtained but also the eigenstresses related with the volume expansion of the inclusion phase can be modeled. However, in the context of FE^2 simulations this procedure is rather expensive since the application of the volumetric strains has to be simulated at each point before the actual loading can be applied. Due to the fact that the above-mentioned eigenstresses are not significant to the macroscopic stresses under loading, here a separated approach is proposed: the first step of applying IVS is considered only to generate the local ferritic yield modification factors which are saved independently of any potential subsequent loading. Then, in the second step of mechanical loading these modification factors are applied to the undeformed microstructure. The main benefit is the reduction of computing time since the IVS has to be performed only once to one microstructure. On the other hand, the eigenstresses obtained from the volume expansion are not included anymore. Note that these eigenstresses are usually removed from the DP steel sheet by a special heat treatment procedure which is why

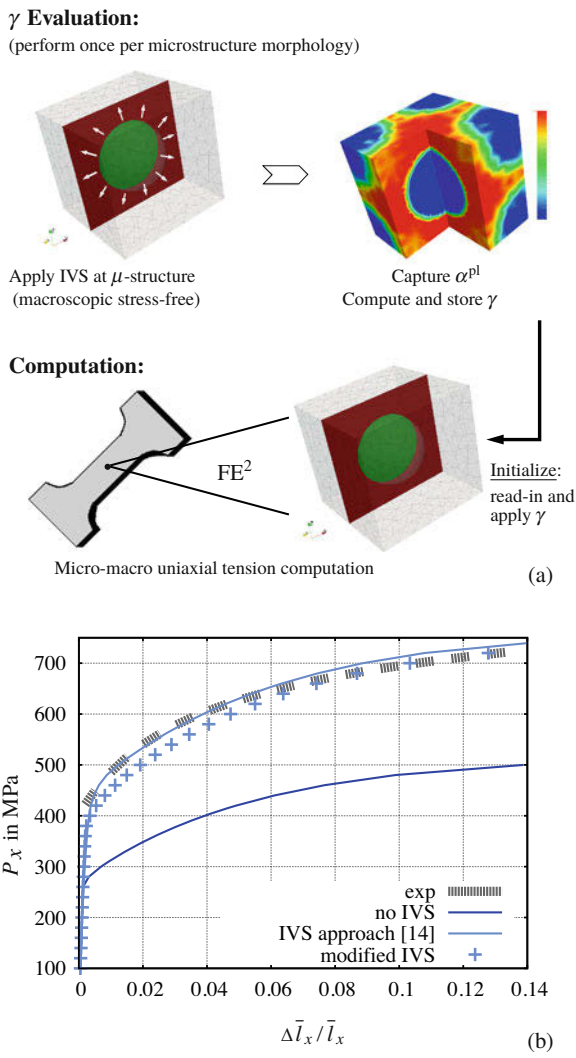


Fig. 20 (**a**) Illustrations of the steps involved in the (modified) initial volumetric strains approach, (**b**) comparison of macroscopic stress-strain curves for FE^2 uniaxial tension calculations for simplified microstructure with spherical inclusion: (i) modified IVS approach, (ii) IVS approach [14], (iii) no IVS approach and (iv) experiment

the absence of these eigenstresses in the numerical simulation may even be more realistic. The scheme is illustrated in Fig. 20a. Furthermore, the macroscopic stress strain curves obtained under uniaxial tension for various IVS considerations are compared against the experiment in Fig. 20b. Here, it can be seen that the proposed modified (separated) IVS scheme performs equivalently to the continuous IVS as given in [14]. However, as seen in Fig. 20b, not using the IVS approach yields a poor accuracy in representing the experimental curve.

Additionally, it is observed that the choice and extent of ferritic hardening has no effect on the resulting factors γ and that the same set can be applied in mechanical loading computations as long as (i) the microstructure, (ii) the amount of martensitic

www.dbooks.org

volume jump considered (i.e. 4%) and (3) the initial yield stress of ferrite (y_0^{iso}), remain unchanged.

The IVS approach has been implemented in FE2TI but has not been applied in the production runs in Sect. 5.

8.2 *Parameter Identification Approach for Ferritic Mixed Hardening*

The incorporation of the mixed hardening in ferrite along with the initial volumetric strains approach necessitates the identification of a new set of material parameters for ferrite, i.e. y_∞^{iso}, η^{iso} and H^{kin}. Here, y_0^{iso} and h^{iso} are assumed to be constant. Since no cyclic stress-strain data is available for the pure ferrite, the ferrite properties need to be adjusted such that the macroscopic response matches well the experiment. Due to the micro-macro nature of the computations required for the resulting inverse problem, this parameter identification problem becomes highly time and computation intensive. Therefore, to accelerate the process a neural network based algorithm is proposed.

As illustrated in Fig. 21, a sufficiently trained neural network takes in eight input values from DP steel experiments and outputs the values for the three parameters to be identified. These input values are as illustrated in Fig. 21b, i.e. seven macroscopic stress values and the Bauschinger factor. The neural network consists of one hidden

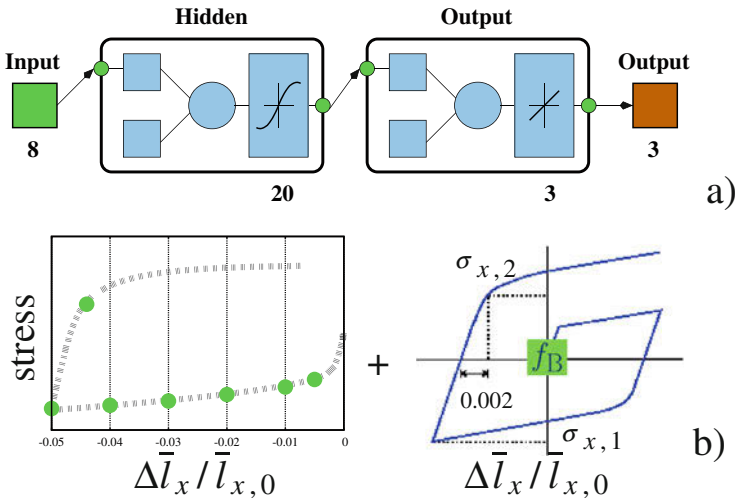


Fig. 21 (**a**) Schematic representation of the neural network with the respective activation functions at various layers, (**b**) the input values for the neural network—7 macroscopic stress-strain values and 1 macroscopic Bauschinger factor

Table 7 Neural network training data range for the identification of ferritic material parameters

	H^{kin}	η^{iso}	y_{∞}^{iso}
Range	0.9–2.1 GPa	15.0–25.0	0.35–0.80 GPa
Nos.	5	4	4

Table 8 (a) Target material parameters for the ferritic phase identified with the trained neural network and (b) Bauschinger factor computed with mixed hardening in ferrite (Sim-mix)

(a)	H^{kin}	η^{iso}	y_{∞}^{iso}	(b)		Exp	Sim-mix
	1.71 GPa	25.6	0.375 GPa		f_B	0.66	0.62

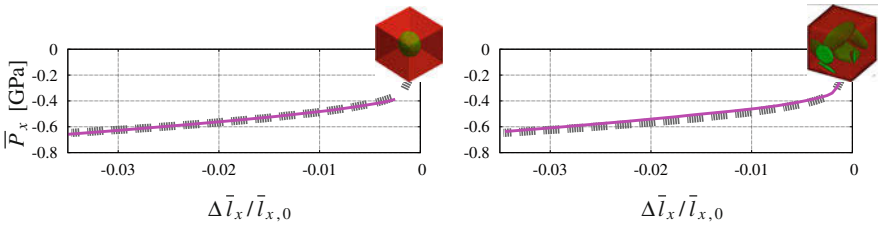


Fig. 22 Stress response comparison between simulation model with mixed hardening for ferrite and the experiment

layer with *tanh* type activation functions and one output layer with linear activation functions. Results from 80 simulations with the choice of target parameters in the ranges mentioned in Table 7 using the simplified microstructure (spherical inclusion in a cuboidal matrix) are used as training data. These simulations with different target parameter combinations can all be executed at once on many core machines to accelerate the process of gathering training data.

Additionally, a good choice of training range helps to ensure a robust prediction of the target parameters. The parameters identified by evaluating this algorithm are given in Table 8a. The macroscopic stresses achieved during compression and the overall Bauschinger factor obtained with these parameters are in Fig. 22 and Table 8b, respectively. These indicate a good match with the experimental observations. Additionally, it is found that the identified material parameters predict a higher pure ferritic yield curve than observed in experiments on the lab-synthesized pure ferrite. It is emphasized that it is in principle challenging to synthesize a pure ferrite which corresponds to the ferrite in the DP steel with respect to similarity in chemical composition and grain size distribution. Therefore, the experimental data regarding the pure ferritic behavior should generally be only considered carefully.

8.3 Quantification of Uncertain Material Properties

The material parameters for the micro-constituents of the DP steel are usually obtained based on experiments on limited numbers of samples. Since the material

www.dbooks.org

behavior of the constituents depends on the production process parameters, which may be non-uniform due to the nature of the process over large batch sizes, the measurements might not accurately represent the complete reality. This holds in particular for specialized laboratory productions of samples only consisting of the pure ferrite phase, which matches the microstructure and chemical composition as closely as possible compared to the ferrite in the DP steel. Due to the fact that these ferrite properties are however believed to strongly influence the properties of the overall DP steel behavior an associated uncertainty quantification analysis was conducted as part of the project. Based on such analysis the sensitivity of the macroscopic stress-strain response of DP steel for modified ferritic properties can be investigated. For the analysis employed here, known probability distributions are assumed for selected ferritic parameters which are (i) y_0^{iso} and y_∞^{iso} and (ii) H^{kin}. It should be noted that varying y_0^{iso} and y_∞^{iso} together for ferrite leads to a change in the height of the ferritic yield curve. For each of the cases, 15,000 samples are randomly constructed to generate Gaussian distributions as input uncertainty distributions for the ferrite parameters y_0^{iso} and H^{kin}; see Fig. 23.

Based on these assumed input distributions of the ferrite parameters the resulting distributions of macroscopic properties are to be computed. Trained neural networks are used here again to evaluate each of these samples and to compute the macroscopic DP steel responses for (i) the yield stress in compression ($\overline{R}_{p0.25}$), (ii) the Bauschinger factor ($\overline{f}_{\text{B}}$) and (iii) the hardening modulus around 5% compression ($\overline{H}_{\text{end}}$).

The output uncertainties in the above mentioned macroscopic measures are then plotted in the sense of their co-relation with the ferritic yield curve and the prescribed kinematic hardening parameter in Figs. 24 and 25 respectively. The correlation between the output macroscopic initial yield stress ($\overline{R}_{\text{p0.25}}$) and the ferritic yield curve as seen in Fig. 24a for the prescribed input, turns out to be a linear relationship. As evident in Fig. 24b the macroscopic Bauschinger factor, changes non-linearly with the ferritic yield curve. However, the overall small variations in the values of the Bauschinger factor suggest that the height of the ferritic yield curve

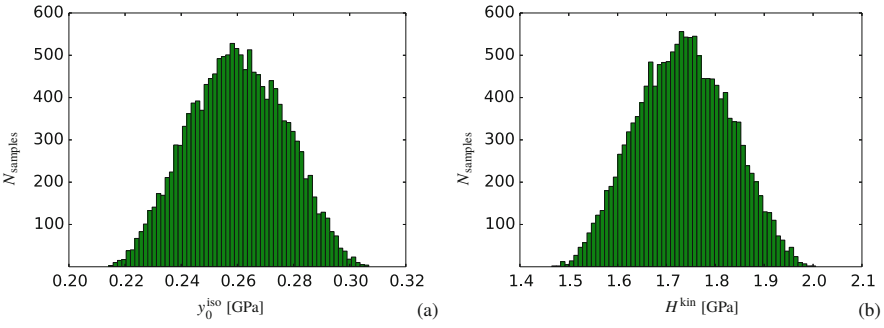


Fig. 23 Input uncertainty distributions for (**a**) variation in y_0^{iso} and (**b**) H^{kin} ferritic material parameters

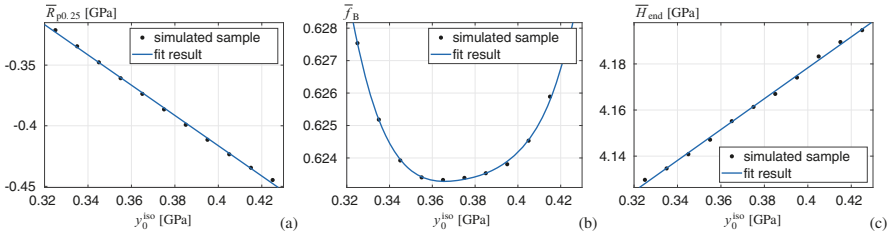


Fig. 24 (**a**) Output uncertainty of (**a**) $\overline{R}_{p0.25}$ stress, (**b**) $\overline{f}_B$ and (**c**) $\overline{H}_{end}$ based on the variation of y_0^{iso}

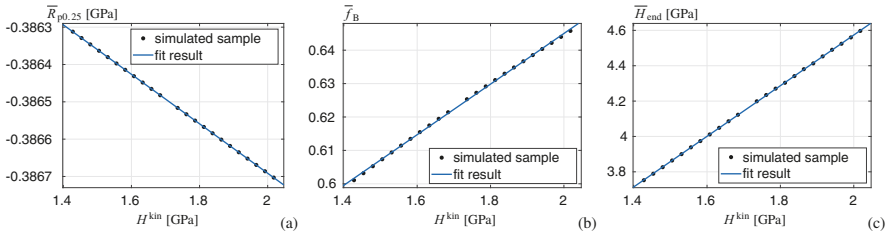


Fig. 25 (**a**) Output uncertainty of (**a**) $\overline{R}_{p0.25}$ stress, (**b**) $\overline{f}_B$ and (**c**) $\overline{H}_{end}$ based on the variation of H^{kin}

only negligibly influences the macroscopic Bauschinger factor. Thus, it appears that it is the overall large difference in yield stress between the ferrite and the martensite rather than moderate changes within the ferrite itself, which is responsible for the relatively large kinematic hardening of DP steel. Fig. 24c indicates a linear relationship of $\overline{H}_{end}$ with the ferritic yield curve height. Again, the values of macroscopic moduli change only insignificantly, which indicates a small sensitivity of macroscopic response due to modifications of the ferrite yield stress.

Now, the influence of the kinematic hardening is investigated. The variation in the above mentioned macroscopic quantities has been considered for a prescribed uncertainty in the linear kinematic hardening modulus H^{kin} of the pure ferrite phase. The results for the macroscopic initial yield stress $\overline{R}_{p0.25}$ are plotted in Fig. 25a, where a linearly reducing correlation is observed with increasing ferritic H^{kin}. As before, the values indicate a negligible change in $\overline{R}_{p0.25}$ with modifications of H^{kin}. This changes significantly for the macroscopic Bauschinger factor, see the results in Fig. 25b, which indicate a strong influence of the macroscopic Bauschinger factor due to modifications of the ferritic kinematic hardening modulus. The relationship is a linearly increasing one. Likewise, an also rather significant, linearly increasing correlation is observed between $\overline{H}_{end}$ and H^{kin}, which has been plotted in Fig. 25c. The results indicate that while the variation in the height of the ferritic yield curve results in a considerable effect on the macroscopic initial yield stress and hardening modulus, it only negligibly influences the Bauschinger factor. Whereas, the ferritic kinematic hardening has a strong influence on the macroscopic response of the DP

www.dbooks.org

steel, especially the macroscopic Bauschinger factor. This further highlights the necessity of employing a mixed hardening based ferritic material model for the micro-macro simulation of relevant DP steel forming problems where effects such as spring-back are of major importance.

Especially for uncertainty quantification problems, where the variation in the microstructure's morphology is considered as source of uncertainty, a high number of different statistically similar volume elements (SSVEs) needs to be simulated. For this purpose, an optimal decomposition approach in the context of a finite cell integration scheme was developed in this project, see [26]. This approach allows for an automated calculation without the necessity to construct a new mesh for each SSVE while keeping the overall computing time even lower than for a conforming (standard) mesh.

8.4 Crystal Plasticity

A better description of certain phenomena, e.g., localization, in crystalline materials can be achieved by explicitly modeling the polycrystalline structure of the material.

Such materials consist of various single crystals with different orientations which interact through the granular interfaces. By directly modeling the plastic behavior of these single crystals, anisotropic yield and complex flow behavior can be captured directly. As pointed out in Sect. 8, this would lead to computationally highly expensive simulations, which can be overcome using approximations of the response of the underlying polycrystal. However, to illustrate the procedure and complexity of incorporating polycrystalline microstructures directly into multiscale simulations, a single crystal plasticity model for face-centered cubic (fcc) crystals at small strains has been implemented, considering an additive decomposition of the small strain tensor into elastic and plastic part $\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}^{\mathrm{e}} + \boldsymbol{\varepsilon}^{\mathrm{p}}$ where $\dot{\boldsymbol{\varepsilon}}^{\mathrm{p}} = \sum_{\delta} \dot{\gamma}^{\delta} \boldsymbol{P}^{\delta}$ directly connects the inelastic behavior in the individual grains to the inherent crystallographic structure through the dependency of the rate of plastic strain on the projected rate of plastic slip $\dot{\gamma}^{\delta}$ summed over all systems δ. Therein, the projection tensor $\boldsymbol{P}^{\delta} = \mathrm{sym}\left(\boldsymbol{s}^{\delta} \otimes \boldsymbol{n}^{\delta}\right)$ is defined based on the orthonormal vectors $\boldsymbol{s}^{\delta} \perp \boldsymbol{n}^{\delta}$, describing the slip system δ. Single crystal plasticity models can be distinguished into rate-independent and rate-dependent models. Algorithms of the former type are typically governed by the issue of non-uniqueness of choice of active slip systems among all possible ones, [3, 16], which adds to the complexity of the material model. Different approaches exist to handle this intrinsic problem by e.g. simple perturbation techniques [74], augmented Lagrangian methods [85] or penalty approaches. Recently, an alternative approach to handle the issue of non-uniqueness among the activity of slip systems has been proposed in [84], which uses Infeasible Primal-Dual Interior Point methods for solving the constraint optimization problem. This method uses barrier functions combined with the given constraints of the problem in order to penalize the approach of the boundary of the feasible domain. In contrast to that, rate-dependent algorithms consider all slip systems to be active and

link the rate of slip $\dot{\gamma}^\delta$ on each system δ directly to the Schmid stress $\tau^\delta = \boldsymbol{\sigma} : \boldsymbol{P}^\delta$. The kinetic law reads

$$\dot{\gamma}^\delta = \dot{\gamma}_0 \left| \frac{\tau^\delta}{g^\delta} \right|^{p-1} \left(\frac{\tau^\delta}{g^\delta} \right) \qquad \text{with} \qquad \dot{g}^\delta = \sum_\beta h^{\delta\beta} \left| \dot{\gamma}^\beta \right|, \tag{7}$$

as, e.g., proposed by [39], where the hardening moduli $h^{\delta\beta}$ depends on the strain-like internal variable A with $\dot{A} = \sum_\delta \left| \dot{\gamma}^\delta \right|$.

8.5 Macroscopic Yield Surface Based on Polycrystalline RVEs

In this section, we use two-scale simulations using crystal plasticity to compute macroscopic yield surfaces. These yield surfaces can then be used in FE2TI simulations without directly incorporating crystal plasticity.

The influence of the microscopic polycrystalline material can be considered to compute the resulting macroscopic anisotropic yield surfaces, as mentioned in Sect. 8 and included in a hierarchical multiscale approach, see [32]. In the following, a microstructure consisting of a polycrystal with multiple grains is considered to model its macroscopic yield behavior. Here, for the computation of macroscopic yield surfaces based on the microscopic behavior of polycrystalline unit cells, the software Neper is used to generate a periodic unit cell with 15 grains. The geometry is meshed using 10-noded tetrahedral finite elements. In order to account for an isotropic orientation distribution of the polycrystalline unit cell, each grain is assigned to a specific orientation following from a geodesic dome. For details, we refer to [82]. With these unit cells, macroscopic yield curves based on macroscopic biaxial loading paths, i.e. $\overline{\sigma}_1 : \overline{\sigma}_2, \overline{\sigma}_3 = 0$, are computed in an FE^2 scheme. The stress-driven simulation requires small time steps, which is amplified by the small time step size required for the rate-dependent formulation of single crystal plasticity. Figure 26 shows the initial yield surface at $\langle \alpha \rangle = 3.3 \cdot 10^{-8}$ as well as the distribution of α inside the unit cell. Since the rate-dependent formulation does not have a distinct yield point and the rate-independent behavior is here modeled with $p = 200$, see Eq. (7), this value of equivalent plastic strains has been arbitrarily chosen by the authors. The evolved macroscopic yield surface based on a polycrystalline unit cell at $\langle \alpha \rangle = 4.7 \cdot 10^{-4}$ and a respective distribution of α is shown in Fig. 26. As pointed out in [11], the initial yield surface forms the shape of a Tresca-type yield criterion, whereas the further evolved yield surface is of typical elliptical Mises-type.

www.dbooks.org

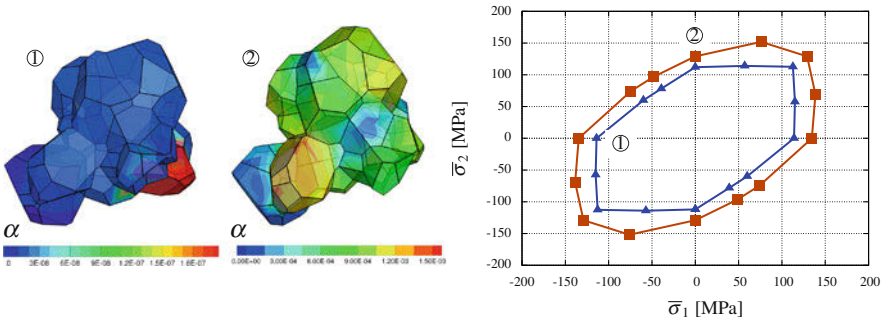


Fig. 26 Polycrystal with 15 grains. Distribution of equivalent plastic strains α for loading path $\overline{\sigma}_1 : \overline{\sigma}_2 = -1 : 0$ at $\alpha_1^* = \langle\alpha\rangle = 3.3 \cdot 10^{-8}$ (left) and for loading path $\overline{\sigma}_1 : \overline{\sigma}_2 = 0 : 1$ at $\alpha_2^* = \langle\alpha\rangle = 4.7 \cdot 10^{-4}$ (center). Right: Associated successive yield surfaces; Tresca-type for α_1^* and von Mises type for α_2^*

8.6 *One-Way Coupled Simulation of Deep-Drawing Using Polycrystalline Unit Cells*

Finally, in this section, we demonstrate a two-scale simulation directly incorporating crystal plasticity on the micro scale. Such simulations are computationally expensive. Only a one-way coupling is used here, and J_2 elasto-plasticity is applied on the macroscale.

In the following, a sheet metal forming process of deep-drawing of a hat-profile, adopted from [7], using an Al-Cu alloy is simulated under consideration of the polycrystalline microstructure in a one-way coupled FE scheme. In Fig. 27, the finite element mesh (165 linear quadrilateral elements) is shown. The interaction between the sheet and the tools is realized with a frictionless penalty contact formulation. The macroscale simulation is carried out using a finite J_2 elasto-plasticity model with isotropic von Mises yield behavior based on an algorithmic setting by [65]. The material parameters, cf. Eq. (5), were fitted to a macroscopic uniaxial tension test with the polycrystalline unit cell used on the microscale leading to $\overline{\kappa}^{\text{Al-Cu}} = 50{,}754$ N/mm^2, $\overline{\mu}^{\text{Al-Cu}} = 23{,}425$ N/mm^2, $\overline{y}_0^{\text{Al-Cu}} = 125$ N/mm^2, $\overline{y}_\infty^{\text{Al-Cu}} = 160$ N/mm^2, $\overline{\eta}^{\text{Al-Cu}} = 750$ and $\overline{h}^{\text{Al-Cu}} = 1$ N/mm^2. The final state of the sheet forming simulation is depicted in Fig. 28 and the distribution of equivalent plastic strain is shown. Throughout the simulation, the deformation gradient $\overline{\boldsymbol{F}}$ is captured at three different positions, marked by $\square$, $\triangle$ and $\bigcirc$ therein, at the top, center and bottom of the sheet, respectively, leading to nine evaluation points in total.

The recorded deformation is applied to a polycrystalline unit cell in a one-way FE coupling. The single crystal plasticity computation is performed at small strains, as described in Sect. 8.4. Thereby, the applied material parameters are taken from [92] with Lamé constant $\lambda = 35{,}104.88$ N/mm^2, shear modulus $\mu = 23{,}427.25$ N/mm^2,

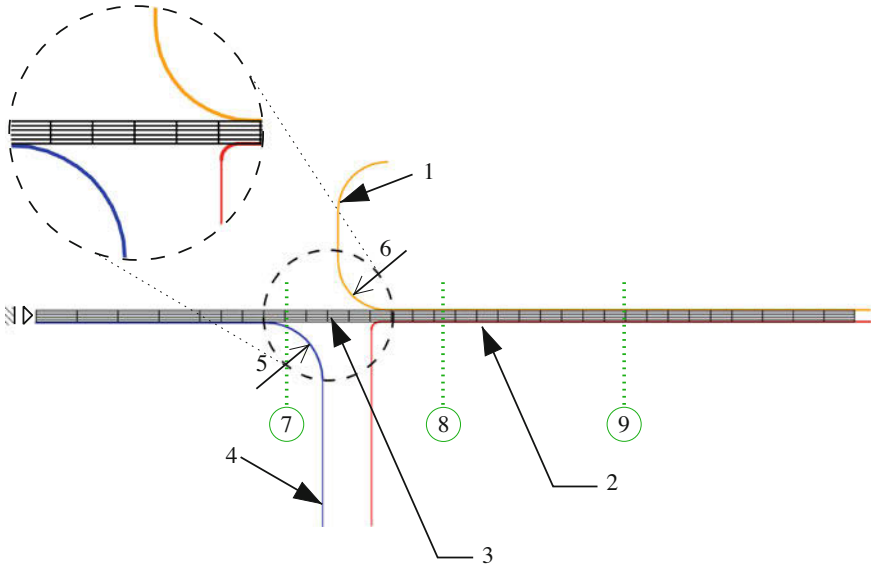


Fig. 27 Discretization of the macroscopic BVP of the deep-drawing of a hat profile under plane strain conditions. 1: drawing die, 2: blank holder, 3: sheet metal (discretized with 5×33 elements), 4: punch, 5: punch radius of 7 mm, and 6: die radius of 6 mm are used. The contact definitions between punch, drawing die, blank holder, and sheet metal are realized using a frictionless penalty formulation. The analyzed RVEs in the macroscopic BVP are located near the punch radius, ⑦, in the vertical section, ⑧, and near the die radius, ⑨, according to the final deformation, cf. 28 (left). The drawing depth of the hat profile is 45.7 mm with a sheet half width of 100 mm and a thickness of 1.4 mm

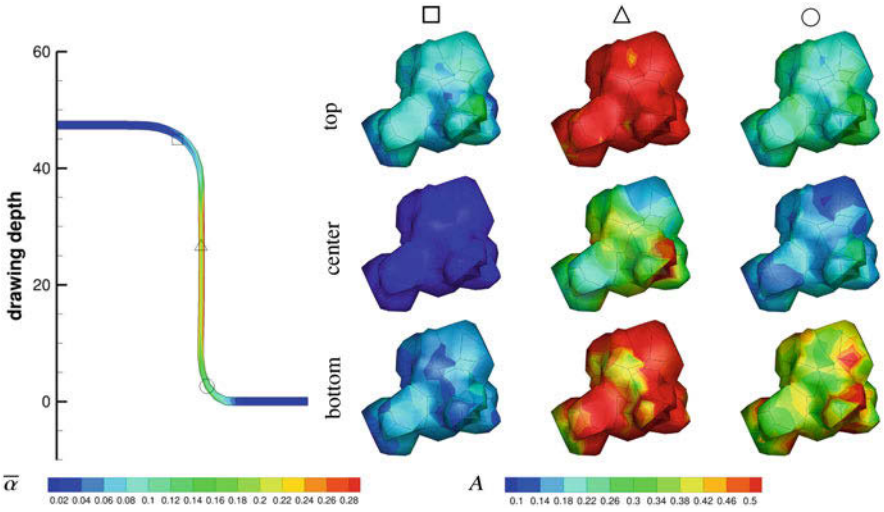


Fig. 28 Distribution of $\overline{\alpha}$ in sheet metal at final deformed state ($t = 50$) and strain-like internal variable A in polycrystalline unit cells at different points

www.dbooks.org

initial slip resistance $\tau_0 = 60.84$ N/mm^2, saturation stress $\tau_\infty = 109.51$ N/mm^2, initial hardening modulus $h_0 = 541.48$ N/mm^2, material rate sensitivity parameter $p = 200$, and reference slip rate $\dot{\gamma}_0 = 1 \cdot 10^{-3}$. The small strain tensor $\overline{\boldsymbol{\varepsilon}}$ is used to transfer the deformation state from the macroscale to the microscale, however, no coupling back from micro- to macroscale is considered.

In Fig. 28, the distribution of equivalent plastic strain $\overline{\alpha}$ in the hat-profile and the distribution of the strain-like internal variable A as a result of the evaluation of the one-way coupled polycrystalline unit cells is shown. Differences between the positions of the polycrystals in the sheet are obvious as well as the nonhomogeneous distribution of A.

9 Conclusion

The vision of the EXASTEEL project is to develop a virtual HPC laboratory allowing for predictive virtual material testing of modern steels. On this path, we have moved forward in several directions: Since the properties of modern dual-phase steels largely stem from their microstructure, homogenization is indispensable to achieve our goals. We therefore have developed and implemented the FE2TI library, a highly scalable software for computational homogenization based on the FE^2 approach (Sects. 2–4). This approach was then used, for the first time, to compute a forming limit diagram for DP600 steel using the JUWELS supercomputer (Sect. 5). Let us remark that the computation of an FLD is already a step beyond the achievements envisaged in the original EXASTEEL-2 proposal. We have also shown scalability of the FE2TI package up to the largest supercomputers currently available, e.g., using more than one million MPI ranks for nonlinear production problems, i.e., using unstructured meshes, elasto-plasticity, and full parallel I/O [64] (Sect. 3). These latter simulations use parallel FETI-DP solvers for the RVE problems and made use of the full JUQUEEN supercomputer.

To move towards full use of the future exascale supercomputers, we have worked on extending the parallel scalability of implicit nonlinear FETI-DP and BDDC domain decomposition solvers (Sect. 6). Scalability to 800,000 parallel tasks was achieved for our nonlinear solvers [54], outside of our parallel FE^2 multiscale context; see Fig. 15. These simulations used the full Mira supercomputer. We have also considered techniques to improve the energy efficiency of our nonlinear domain decomposition solvers (Sect. 6.1.1). Careful performance analysis and engineering was applied to the FE2TI software building blocks, e.g., for the performance engineering of the sparse triangular solves of the PARDISO sparse direct solver (Sect. 7).

For the modeling, considering initial volumetric strains, resulting from the complex steel production process, has shown to be of interest; therefore, an efficient algorithmic approach to IVS was proposed (Sects. 8.1 and 8.2). This IVS approach has been implemented in FE2TI. Further improvements in our modeling

may be achieved by incorporating effects from crystal plasticity (Sect. 8.4). An approach to fit macroscopic yield surfaces to crystal plasticity simulations was presented (Sect. 8.5). The resulting yield surfaces can be used in FE2TI without using an explicit coupling with crystal plasticity simulations. However, we have also demonstrated a two-scale simulation using a one-way coupling with crystal plasticity (Sect. 8.6).

For the quantitatively predictive simulations envisaged in this project, several improvements are planned for the future. First, realistic material models reproducing the physics on the microscale are important. Different advanced approaches beyond the ones considered so far may be of interest, e.g., based on the techniques described in Sect. 8. Second, for the computation of the FLD, the exploitation of the symmetry of the Nakajima specimen has to be reviewed and, especially for strongly anisotropic microstructures, simulations using the full geometry have to be performed for all specimen. Third, a validation with experiments for steels other than DP600 will be necessary. Finally, once exascale supercomputers will be available, predictive virtual steel simulations at the exascale will leverage the combined parallelism of the FE^2 algorithm and of the parallel nonlinear domain decomposition solvers.

Acknowledgments This work was supported by the German Research Foundation (DFG) through the Priority Programme 1648 "Software for Exascale Computing" (SPPEXA), DFG project 230723766 and by the Swiss National Science Foundation (SNSF), project 200021E-162296. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer **JUWELS** and its predecessor **JUQUEEN** at Jülich Supercomputing Centre (JSC). Computational resource of Fujitsu PRIMERGY CX600M1/CX1640M1(**Oakforest-PACS**) was awarded by "Large-scale HPC Challenge" project, Joint Center for Advanced High Performance Computing (JCAHPC). The authors also gratefully acknowledge the computing time granted by the Center for Computational Sciences and Simulation (CCSS) of the Universität of Duisburg-Essen and provided on the supercomputer **magnitUDE** (DFG grants INST 20876/209-1 FUGG, INST 20876/243-1 FUGG) at the Zentrum für Informations- und Mediendienste (ZIM) as well as the compute resources on **Meggie** and support provided by the Erlangen Regional Computing Center (RRZE). This research used the resources **Mira** and **Theta** of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. Furthermore, the access to the computing cluster **Taurus** provided by the ZIH, Technical University of Dresden and the computing cluster **SD Cluster** provided by the Faculty of Civil and Environmental Engineering, Ruhr-University Bochum are also gratefully acknowledged.

References

1. Alappat, C., Hager, G., Schenk, O., Thies, J., Basermann, A., Bishop, A., Fehske, H., Wellein, G.: A recursive algebraic coloring technique for hardware-efficient symmetric sparse matrix-vector multiplication. ACM Trans. Parallel Comput. (2020, accepted). arXiv e-prints, ArXiv:1907.06487
2. Amestoy, P., Duff, I., L'Excellent, J.Y., Koster, J.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. **23**(1), 15–41 (2002)

www.dbooks.org

3. Anand, L., Kothari, M.: A computational procedure for rate-independent crystal plasticity. J. Mech. Phys. Solids **44**, 525–558 (1996)
4. Announcement of the DFG Priority Programme 'Software for Exascale Computing' (SPP 1648) (2011). https://www.dfg.de/foerderung/info_wissenschaft/2011/info_wissenschaft_11_59/index.html
5. Baker, A., Klawonn, A., Kolev, T., Lanser, M., Rheinbach, O., Yang, U.: Scalability of classical algebraic multigrid for elasticity to half a million parallel tasks. Lect. Notes Comput. Sci. Eng. **113**, 113–140 (2016)
6. Balay, S., Abhyankar, S., Adams, M., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W., Karpeyev, D., Kaushik, D., Knepley, M., May, D., McInnes, L.C., Mills, R.T., Munson, T., Rupp, K., Sanan, P., Smith, B., Zampini, S., Zhang, H.: PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.11, Argonne National Laboratory (2019). https://www.mcs.anl.gov/petsc
7. Balzani, D., Brands, D., Schröder, J.: Construction of statistically similar representative volume elements. In: Schröder, J., Hackl, K. (eds.) Plasticity and Beyond - Microstructures, Crystal-Plasticity and Phase Transitions (CISM Lecture Notes 550), pp. 355–412. Springer, Berlin (2014)
8. Balzani, D., Scheunemann, L., Brands, D., Schröder, J.: Construction of two- and three-dimensional statistically similar RVEs for coupled micro-macro simulations. Comput. Mech. **54**, 1269–1284 (2014)
9. Balzani, D., Gandhi, A., Klawonn, A., Lanser, M., Rheinbach, O., Schröder, J.: One-way and fully-coupled FE2 methods for heterogeneous elasticity and plasticity problems: parallel scalability and an application to thermo-elastoplasticity of dual-phase steels. Lect. Notes Comput. Sci. Eng. **113**, 91–112 (2016). https://doi.org/10.1007/978-3-319-40528-5_5. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84989948238&doi=10.1007%2f978-3-319-40528-5_5&partnerID=40&md5=5c4efedccb3dab06ef11fcc5b2a61b2e
10. Bhardwaj, M., Pierson, K., Reese, G., Walsh, T., Day, D., Alvin, K., Peery, J., Farhat, C., Lesoinne, M.: Salinas: a scalable software for high-performance structural and solid mechanics simulations. In: Proceedings of the 2002 ACM/IEEE Conference on Supercomputing, SC '02, pp. 1–19. IEEE Computer Society Press, Los Alamitos (2002). http://dl.acm.org/citation.cfm?id=762761.762790
11. Böhlke, T., Kraska, M., Bertram, A.: Simulation der einfachen Scherung einer polykristallinen Aluminiumprobe. Tech. Mech., Sonderheft 47–54 (1997)
12. Bollhöfer, M., Schenk, O., Janalík, R., Hamm, S., Gullapalli, K.: State-of-The-Art Sparse Direct Solvers. Parallel Algorithms in Computational Science & Engineering - Parallelism as Enabling Technology in CSE Applications, Birkhauser (2019). ArXiv: 1907.05309
13. Bordeu, F., Boucard, P., Gosselet, P.: Balancing domain decomposition with nonlinear relocalization: parallel implementation for laminates. Civil-Comp Proc. **90** (2009). https://www.scopus.com/inward/record.uri?eid=2-s2.0-84894120361&partnerID=40&md5=fa707dc23f7475e5878d87c48103e443
14. Brands, D., Balzani, D., Scheunemann, L., Schröder, J., Richter, H., Raabe, D.: Computational modeling of dual-phase steels based on representative three-dimensional microstructures obtained from EBSD data. Arch. Appl. Mech. 1–24 (2016). https://doi.org/10.1007/s00419-015-1044-1
15. Brune, P., Knepley, M., Smith, B., Tu, X.: Composing scalable nonlinear algebraic solvers. SIAM Rev. **57**(4), 535–565 (2015). https://doi.org/10.1137/130936725
16. Busso, E., Cailletaud, G.: On the selection of active slip systems in crystal plasticity. Int. J. Plast. **21**, 2212–2231 (2005)
17. Cai, X.C., Keyes, D.: Nonlinearly preconditioned inexact Newton algorithms. SIAM J. Sci. Comput. **24**(1), 183–200 (2003). https://doi.org/10.1137/S106482750037620X. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0037248934&doi=10.1137%2fS106482750037620X&partnerID=40&md5=b222326c33fbbec11255e3de162d54d4

18. Cai, X.C., Keyes, D., Marcinkowski, L.: Non-linear additive Schwarz preconditioners and application in computational fluid dynamics. Int. J. Numer. Methods Fluids **40**(12), 1463–1470 (2002). https://doi.org/10.1002/fld.404. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0037203227&doi=10.1002%2ffld.404&partnerID=40&md5=bdbadff518af94aae714206b017180d6
19. Cockcroft, M., Latham, D.: Ductility and the workability of metals. J. Inst. Met. **48**, 33–39 (1968)
20. Cros, J.M.: A preconditioner for the Schur complement domain decomposition method. In: Herrera, O.W.I., Keyes, D., Yates, R. (eds.) Domain Decomposition Methods in Science and Engineering, pp. 373–380. National Autonomous University of Mexico (UNAM), Mexico City (2003). ISBN 970-32-0859-2. Proceedings of the 14th International Conference on Domain Decomposition Methods in Science and Engineering. http://www.ddm.org/DD14
21. Daly, J.: A model for predicting the optimum checkpoint interval for restart dumps. In: Sloot, P.M.A., Abramson, D., Bogdanov, A.V., Gorbachev, Y.E., Dongarra, J.J., Zomaya, A.Y. (eds) Computational Science—ICCS 2003 (ICCS 2003). Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 2660, pp. 3–12. Springer, Berlin (2003). https://doi.org/10.1007/3-540-44864-0_1
22. Davis, T.: A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. ACM Trans. Math. Softw. **30**(2), 165–195 (2004)
23. De Coninck, A., De Baets, B., Kourounis, D., Verbosio, F., Schenk, O., Maenhout, S., Fostier, J.: Needles: toward large-scale genomic prediction with marker-by-environment interaction. Genetics **203**(1), 543–555 (2016). https://doi.org/10.1534/genetics.115.179887. http://www.genetics.org/content/203/1/543
24. Dohrmann, C.: A preconditioner for substructuring based on constrained energy minimization. SIAM J. Sci. Comput. **25**(1), 246–258 (2003). https://doi.org/10.1137/S1064827502412887
25. Falgout, R., Jones, J., Yang, U.: The design and implementation of hypre, a library of parallel high performance preconditioners. Lect. Notes Comput. Sci. Eng. **51**, 267–294 (2006)
26. Fangye, Y., Miska, N., Balzani, D.: Automated simulation of voxel-based microstructures based on enhanced finite cell approach. Arch. Appl. Mech. 2020, accepted
27. Farhat, C., Lesoinne, M., Pierson, K.: A scalable dual-primal domain decomposition method. Numer. Linear Algebra Appl. **7**(7-8), 687–714 (2000). https://doi.org/10.1002/1099-1506(200010/12)7:7/8<687::AID-NLA219>3.0.CO;2-S. Preconditioning techniques for large sparse matrix problems in industrial applications (Minneapolis, MN, 1999)
28. Farhat, C., Lesoinne, M., LeTallec, P., Pierson, K., Rixen, D.: FETI-DP: a dual-primal unified FETI method. I. A faster alternative to the two-level FETI method. Int. J. Numer. Methods Eng. **50**(7), 1523–1544 (2001). https://doi.org/10.1002/nme.76
29. Feyel, F.: Multiscale FE^2 elastoviscoplastic analysis of composite structures. Comput. Mater. Sci. **16**(1-4), 344–354 (1999). https://www.scopus.com/inward/record.uri?eid=2-s2.0-0042142743&partnerID=40&md5=c5b0208bdde06570bc8a5898347b88f1
30. Feyel, F.: A multilevel finite element method (FE^2) to describe the response of highly non-linear structures using generalized continua. Comput. Methods Appl. Mech. Eng. **192**(28-30), 3233–3244 (2003). https://doi.org/10.1016/S0045-7825(03)00348-7. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0043127126&doi=10.1016%2fS0045-7825%2803%2900348-7&partnerID=40&md5=75de4641820478ccd41e5aa8e8d4d7f2
31. Feyel, F., Chaboche, J.L.: FE^2 multiscale approach for modelling the elastoviscoplastic behaviour of long fibre SiC/Ti composite materials. Comput. Methods Appl. Mech. Eng. **183**(3-4), 309–330 (2000). https://doi.org/10.1016/S0045-7825(99)00224-8. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0034677856&doi=10.1016%2fS0045-7825%2899%2900224-8&partnerID=40&md5=77470c678941b0c2ade865a2e31afecb
32. Gawad, J., Van Bael, A., Eyckens, P., Samaey, G., Van Houtte, P., Roose, D.: Hierarchical multi-scale modeling of texture induced plastic anisotropy in sheet forming. Comput. Mater. Sci. **66**, 65–83 (2013)

www.dbooks.org

33. Geers, M., Kouznetsova, V., Matouš, K., Yvonnet, J.: Homogenization Methods and Multiscale Modeling: Nonlinear Problems, pp. 1–34. American Cancer Society (2017). https://doi.org/10.1002/9781119176817.ecm2107. https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119176817.ecm2107
34. Geuzaine, C., Remacle, J.F.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int. J. Numer. Methods Eng. **79**(11), 1309–1331 (2009)
35. Groß, C.: A unifying theory for nonlinear additively and multiplicatively preconditioned globalization strategies: convergence results and examples from the field of nonlinear elastostatics and elastodynamics. Ph.D. thesis, Rheinische Friedrich-Wilhelm Universität Bonn (2009). Deutsche Nationalbibliothek https://www.deutsche-digitale-bibliothek.de/item/PCLVYPVW5OCPUOTIKRKTMSHMFSNWEFPL
36. Groß, C., Krause, R.: A generalized recursive trust-region approach - nonlinear multiplicatively preconditioned trust-region methods and applications. Technical report 2010-09, Institute of Computational Science, Universita della Svizzeria italiana (2010)
37. Groß, C., Krause, R.: On the globalization of ASPIN employing trust-region control strategies - convergence analysis and numerical examples. Technical report 2011-03, Institute of Computational Science, Universita della Svizzeria italiana (2011)
38. Henson, V., Yang, U.: BoomerAMG: a parallel algebraic multigrid solver and preconditioner. Appl. Numer. Math. **41**(1), 155–177 (2002). https://doi.org/10.1016/S0168-9274(01)00115-5. Developments and trends in iterative methods for large systems of equations—in memoriam Rüdiger Weiss (Lausanne, 2000)
39. Hutchinson, J.: Bounds and self-consistent estimates for creep of polycrystalline materials. Proc. R. Soc. Lond. A. **348**, 101–127 (1976)
40. Hwang, F.N., Cai, X.C.: Improving robustness and parallel scalability of Newton method through nonlinear preconditioning. Lect. Notes Comput. Sci. Eng. **40**, 201–208 (2005). https://www.scopus.com/inward/record.uri?eid=2-s2.0-33751093392&partnerID=40&md5=214fdd3b1e445839e0232863a17b3076
41. Hwang, F.N., Cai, X.C.: A class of parallel two-level nonlinear Schwarz preconditioned inexact Newton algorithms. Comput. Methods Appl. Mech. Eng. **196**(8), 1603–1611 (2007). https://doi.org/10.1016/j.cma.2006.03.019. https://www.scopus.com/inward/record.uri?eid=2-s2.0-33751073924&doi=10.1016%2fj.cma.2006.03.019&partnerID=40&md5=4c60a5f767eb41542ab0a553a1069bf4
42. Hwang, F.N., Su, Y.C., Cai, X.C.: A parallel adaptive nonlinear elimination preconditioned inexact Newton method for transonic full potential equation. Comput. Fluids **110**, 96–107 (2015). https://doi.org/10.1016/j.compfluid.2014.04.005
43. Jocham, D.: Bestimmung der lokalen Einschnürung nach linearer und nichtlinearer Umformhistorie sowie Ermittlung dehnungs- und geschwindigkeitsabhängiger Materialkennwerte. Ph.D. thesis, Technische Universität München (2018)
44. Jülich Supercomputing Centre: JUQUEEN: IBM Blue Gene/Q Supercomputer System at the Jülich Supercomputing Centre. J. Large-Scale Res. Facil. **1**(A1) (2015). http://dx.doi.org/10.17815/jlsrf-1-18
45. Jülich Supercomputing Centre: JUWELS: Modular Tier-0/1 Supercomputer at the Jülich Supercomputing Centre. J. Large-Scale Res. Facil. **5**(A135) (2019). http://dx.doi.org/10.17815/jlsrf-5-171
46. Klawonn, A., Rheinbach, O.: Inexact FETI-DP methods. Int. J. Numer. Methods Eng. **69**(2), 284–307 (2007). https://doi.org/10.1002/nme.1758
47. Klawonn, A., Rheinbach, O.: Robust FETI-DP methods for heterogeneous three dimensional elasticity problems. Comput. Methods Appl. Mech. Eng. **196**(8), 1400–1414 (2007). https://doi.org/10.1016/j.cma.2006.03.023
48. Klawonn, A., Rheinbach, O.: Highly scalable parallel domain decomposition methods with an application to biomechanics. ZAMM Z. Angew. Math. Mech. **90**(1), 5–32 (2010). https://doi.org/10.1002/zamm.200900329
49. Klawonn, A., Widlund, O.: Dual-primal FETI methods for linear elasticity. Commun. Pure Appl. Math. **59**(11), 1523–1572 (2006). https://doi.org/10.1002/cpa.20156

50. Klawonn, A., Widlund, O., Dryja, M.: Dual-primal FETI methods for three-dimensional elliptic problems with heterogeneous coefficients. SIAM J. Numer. Anal. **40**, 159–179 (2002). https://doi.org/10.1137/S0036142901388081
51. Klawonn, A., Lanser, M., Rheinbach, O.: Nonlinear FETI-DP and BDDC methods. SIAM J. Sci. Comput. **36**(2), A737–A765 (2014). https://doi.org/10.1137/130920563. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84899626259&doi=10.1137%2f130920563&partnerID=40&md5=d92241d0098f77167704c1b2cb5aae85
52. Klawonn, A., Lanser, M., Rheinbach, O.: FE2TI (ex_nl/fe2) EXASTEEL - Bridging scales for multiphase steels (2015). https://www.swmath.org/software/13908; see also the report on the JUQUEEN Extreme Scaling Workshop 2015: http://hdl.handle.net/2128/8435
53. Klawonn, A., Lanser, M., Rheinbach, O.: Juqueen Extreme Scaling Workshop 2015. Technical Report FZJ-JSC-IB-2015-01 (2015). Brömmel, D., Frings, W., Wylie, B.J.N. (eds.), http://hdl.handle.net/2128/8435
54. Klawonn, A., Lanser, M., Rheinbach, O.: Toward extremely scalable nonlinear domain decomposition methods for elliptic partial differential equations. SIAM J. Sci. Comput. **37**(6), C667–C696 (2015). https://doi.org/10.1137/140997907
55. Klawonn, A., Lanser, M., Rheinbach, O., Stengel, H., Wellein, G.: Hybrid MPI/OpenMP parallelization in FETI-DP methods. In: Mehl, M., Bischoff, M., Schäfer, M. (eds.) Recent Trends in Computational Engineering - CE2014: Optimization, Uncertainty, Parallel Algorithms, Coupled and Complex Problems, pp. 67–84. Springer International Publishing, Cham (2015). https://doi.org/10.1007/978-3-319-22997-3_4
56. Klawonn, A., Lanser, M., Rheinbach, O.: A highly scalable implementation of inexact nonlinear FETI-DP without sparse direct solvers. In: Karasözen, B., Manguoğlu, M., Tezer-Sezgin, M., Göktepe, S., Uğur, Ö. (eds.) Numerical Mathematics and Advanced Applications ENUMATH 2015, pp. 255–264. Springer International Publishing, Cham (2016)
57. Klawonn, A., Lanser, M., Rheinbach, O.: FE2TI: computational scale bridging for dual-phase steels. Adv. Parallel Comput. **27**, 797–806 (2016). https://doi.org/10.3233/978-1-61499-621-7-797. https://www.scopus.com/inward/record.uri?eid=2-s2.0-84969913450&doi=10.3233%2f978-1-61499-621-7-797&partnerID=40&md5=ae60853a9101d7bad22126f1481cced4
58. Klawonn, A., Lanser, M., Rheinbach, O.: Juqueen Extreme Scaling Workshop 2016. Technical Report FZJ-JSC-IB-2016-01 (2016). Brömmel, D., Frings, W., Wylie, B.J.N. (eds.), http://hdl.handle.net/2128/9990
59. Klawonn, A., Lanser, M., Rheinbach, O., Uran, M.: Nonlinear FETI-DP and BDDC methods: a unified framework and parallel results. SIAM J. Sci. Comput. **39**(6), C417–C451 (2017). https://doi.org/10.1137/16M1102495
60. Klawonn, A., Lanser, M., Rheinbach, O.: Nonlinear BDDC methods with approximate solvers. Electron. Trans. Numer. Anal. **49**, 244–273 (2018)
61. Klawonn, A., Lanser, M., Rheinbach, O.: Using algebraic multigrid in inexact BDDC domain decomposition methods. In: Bjørstad, P., Brenner, S., Halpern, L., Kim, H., Kornhuber, R., Rahman, T., Widlund, O. (eds.) Domain Decomposition Methods in Science and Engineering XXIV, pp. 425–433. Springer, Cham (2018)
62. Klawonn, A., Lanser, M., Rheinbach, O., Uran, M.: On the accuracy of the inner Newton iteration in nonlinear domain decomposition. In: Bjørstad, P., Brenner, S., Halpern, L., Kim, H., Kornhuber, R., Rahman, T., Widlund, O. (eds.) Domain Decomposition Methods in Science and Engineering XXIV, pp. 435–443. Springer International Publishing, Cham (2018)
63. Klawonn, A., Lanser, M., Rheinbach, O., Wellein, G., Wittmann, M.: Energy efficiency of nonlinear domain decomposition methods. Technical Report, Universität zu Köln (2018). https://kups.ub.uni-koeln.de/8654/
64. Klawonn, A., Köhler, S., Lanser, M., Rheinbach, O.: Computational homogenization with million-way parallelism using domain decomposition methods. Comput. Mech. **65**(1), 1–22 (2020). http://dx.doi.org/10.1007/s00466-019-01749-5
65. Klinkel, S.: Theorie und Numerik eines Volume-Schalen-Elementes bei finiten elastischen und plastischen Verzerrungen. Ph.D. thesis, Universität zu Karlsruhe (2000)

www.dbooks.org

66. Knoll, D., Keyes, D.: Jacobian-free Newton-Krylov methods: a survey of approaches and applications. J. Comput. Phys. **193**(2), 357–397 (2004). https://doi.org/10.1016/j.jcp.2003.08.010. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0348198275&doi=10.1016%2fj.jcp.2003.08.010&partnerID=40&md5=3555dff5a4109f37815938eaf876d64c
67. Konyukhov, A., Schweizerhof, K.: Contact formulation via a velocity description allowing efficiency improvements in frictionless contact analysis. Comp. Mech. **33**(3), 165–173 (2004)
68. Konyukhov, A., Schweizerhof, K.: On some aspects for contact with rigid surfaces: surface-to-rigid surface and curves-to-rigid surface algorithms. Comput. Methods Appl. Mech. Eng. **283**, 74–105 (2014)
69. Kourounis, D., Fuchs, A., Schenk, O.: Towards the next generation of multiperiod optimal power flow solvers. IEEE Trans. Power Syst. **PP**(99), 1–10 (2018). https://doi.org/10.1109/TPWRS.2017.2789187
70. Kouznetsova, V., Brekelmans, W., Baaijens, F.: Approach to micro-macro modeling of heterogeneous materials. Comput. Mech. **27**(1), 37–48 (2001). https://doi.org/10.1007/s004660000212. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0035088146&doi=10.1007%2fs004660000212&partnerID=40&md5=544bd0552f2392765175af3e69542b91
71. Li, J., Widlund, O.: FETI–DP, BDDC, and Block Cholesky methods. Int. J. Numer. Methods Eng. **66**(2), 250–271 (2006)
72. Mandel, J., Dohrmann, C.: Convergence of a balancing domain decomposition by constraints and energy minimization. Numer. Linear Algebra Appl. **10**, 639–659 (2003)
73. Mandel, J., Dohrmann, C., Tezaur, R.: An algebraic theory for primal and dual substructuring methods by constraints. Appl. Numer. Math. **54**(2), 167–193 (2005). https://doi.org/10.1016/j.apnum.2004.09.022. https://www.scopus.com/inward/record.uri?eid=2-s2.0-19044366698&doi=10.1016%2fj.apnum.2004.09.022&partnerID=40&md5=fd842620bb07336ba64bc9dc2168d184
74. Miehe, C., Schröder, J.: A comparative study of stress update algorithms for rate independent and rate dependent crystal plasticity. Int. J. Numer. Methods Eng. **50**, 273–298 (2001)
75. Miehe, C., Schröder, J., Schotte, J.: Computational homogenization analysis in finite plasticity Simulation of texture development in polycrystalline materials. Comput. Methods Appl. Mech. Eng. **171**(3), 387–418 (1999)
76. Nocedal, J., Wright, S.: Numerical Optimization. Springer Series in Operations Research and Financial Engineering, 2nd edn. Springer, New York (2006)
77. Norm DIN EN ISO 12004-2:2008: Metallic materials – sheet and strip – determination of forming-limit curves – part 2: determination of forming-limit curves in the laboratory (2008)
78. Pebrel, J., Rey, C., Gosselet, P.: A nonlinear dual-domain decomposition method: application to structural problems with damage. Int. J. Multiscale Comput. Eng. **6**(3), 251–262 (2008). https://doi.org/10.1615/IntJMultCompEng.v6.i3.50. https://www.scopus.com/inward/record.uri?eid=2-s2.0-54949155511&doi=10.1615%2fIntJMultCompEng.v6.i3.50&partnerID=40&md5=b1dd3dc619dd458bc8b7b4070690731d
79. Prüger, S., Gandhi, A., Balzani, D.: Influence of microstructure morphology on multi-scale modeling of low-alloyed TRIP-steels. Comput. Eng. **35**(2), 499–528 (2017)
80. Rüde, U., Willcox, K., McInnes, L., Sterck, H.: Research and education in computational science and engineering. SIAM Rev. **60**(3), 707–754 (2018). https://doi.org/10.1137/16M1096840
81. Schenk, O., Gärtner, K.: Two-level dynamic scheduling in PARDISO: improved scalability on shared memory multiprocessing systems. Parallel Comput. **28**(2), 187–197 (2002)
82. Scheunemann, L.: Scale-bridging of elasto-plastic microstructures using statistically similar representative volume elements. Ph.D. thesis, Department Civil Engineering, Faculty of Engineering, University Duisburg-Essen (2017)
83. Scheunemann, L., Balzani, D., Brands, D., Schröder, J.: Design of 3D statistically similar representative volume elements based on Minkowski functionals. Mech. Mater. **90**, 185–201 (2015)

84. Scheunemann, L., Nigro, P., Schröder, J., Pimenta, P.: A novel algorithm for rate independent small strain crystal plasticity based on the infeasible primal-dual interior point method. Int. J. Plast. **124**, 1–19 (2020). https://doi.org/10.1016/j.ijplas.2019.07.020
85. Schmidt-Baldassari, M.: Numerical concepts for rate-independent single crystal Plasticity. Comput. Methods Appl. Mech. Eng. **192**, 1261–1280 (2003)
86. Schröder, J.: Homogenisierungsmethoden der nichtlinearen Kontinuumsmechanik unter Beachtung von Stabilitätsproblemen. Habilititationsschrift, Universität Stuttgart (2000). Bericht aus der Forschungsreihe des Instituts für Mechanik (Bauwesen), Lehrstuhl I
87. Schröder, J.: A numerical two-scale homogenization scheme: the FE^2-method. In: Schröder, K.H.J. (ed.) Plasticity and Beyond - Microstructure, Crystal Plasticity and Phase Transitions, CISM International Centre for Mechanical Sciences, vol. 550, pp. 1–64. Springer, Berlin (2014)
88. Shahzad, F., Thies, J., Kreutzer, M., Zeiser, T., Hager, G., Wellein, G.: CRAFT: a library for easier application-level checkpoint/restart and automatic fault tolerance. IEEE Trans. Parallel Distrib. Syst. (2018). https://doi.org/10.1109/tpds.2018.2866794
89. Simo, J.: Algorithms for static and dynamic multiplicative plasticity that preserve the classical return mapping schemes of the infinitesimal theory. Comput. Methods Appl. Mech. Eng. **99**, 61–112 (1992)
90. Simo, J., Hughes, J.: Computational Inelasticity. Interdisciplinary Applied Mechanics - Mechanics and Materials, vol. 7. Springer, New York (1998)
91. Smit, R., Brekelmans, W., Meijer, H.: Prediction of the mechanical behavior of nonlinear heterogeneous systems by multi-level finite element modeling. Comput. Methods Appl. Mech. Eng. **155**(1-2), 181–192 (1998). https://doi.org/10.1016/S0045-7825(97)00139-4. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0032026085&doi=10.1016%2fS0045-7825%2897%2900139-4&partnerID=40&md5=8b9c27800c66a206d2c5352ace9bd78f
92. Steinmann, P., Stein, E.: On the numerical treatment and analysis of finite deformation ductile single crystal plasticity. Comput. Methods Appl. Mech. Eng. **129**, 235–254 (1996)
93. Tarigopula, V., Hopperstad, O., Langseth, M., Clausen, A., Hild, F., Lademo, O.G., Eriksson, M.: A study of large plastic deformations in dual phase steel using digital image correlation and FE analysis. Exp. Mech. **48**(2), 181–196 (2008). https://doi.org/10.1007/s11340-007-9066-4
94. Treibig, J., Hager, G., Wellein, G.: LIKWID: a lightweight performance-oriented tool suite for x86 multicore environments. In: Proceedings of the 2010 39th International Conference on Parallel Processing Workshops, ICPPW '10, pp. 207–216. IEEE Computer Society, Washington (2010). http://dx.doi.org/10.1109/ICPPW.2010.38
95. Uran, M.: High-performance computing two-scale finite element simulations of a contact problem using computational homogenization. Phd thesis, Universität zu Köln (2020)
96. Verbosio, F., Coninck, A.D., Kourounis, D., Schenk, O.: Enhancing the scalability of selected inversion factorization algorithms in genomic prediction. J. Comput. Sci. **22**(Supplement C), 99–108 (2017). https://doi.org/10.1016/j.jocs.2017.08.013
97. Volk, W., Hora, P.: New algorithm for a robust user-independent evaluation of beginning instability for the experimental FLC determination. Int. J. Mater. Form. **4**(3), 339–346 (2011)
98. Wittmann, M., Hager, G., Janalik, R., Lanser, M., Klawonn, A., Rheinbach, O., Schenk, O., Wellein, G.: Multicore performance engineering of sparse triangular solves using a modified roofline model. In: 2018 30th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), pp. 233–241 (2018). https://doi.org/10.1109/CAHPC.2018.8645938
99. Wriggers, P.: Computational Contact Mechanics. Wiley, Chichester (2002)

www.dbooks.org

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



ExaStencils: Advanced Multigrid Solver Generation



Christian Lengauer, Sven Apel, Matthias Bolten, Shigeru Chiba, Ulrich Rüde, Jürgen Teich, Armin Größlinger, Frank Hannig, Harald Köstler, Lisa Claus, Alexander Grebhahn, Stefan Groth, Stefan Kronawitter, Sebastian Kuckuk, Hannah Rittich, Christian Schmitt, and Jonas Schmitt

Abstract Present-day stencil codes are implemented in general-purpose programming languages, such as Fortran, C, or Java, Python or derivates thereof, and harnesses for parallelism, such as OpenMP, OpenCL or MPI. Project ExaStencils pursued a domain-specific approach with a language, called ExaSlang, that is stratified into four layers of abstraction, the most abstract being the formulation in continuous mathematics and the most concrete a full, automatically generated implementation. At every layer, the corresponding language expresses not only computational directives but also domain knowledge of the problem and platform to be leveraged for optimization. We describe the approach, the software technology

C. Lengauer (✉) · A. Größlinger · A. Grebhahn · S. Kronawitter
University of Passau, Passau, Germany
e-mail: christian.lengauer@uni-passau.de; armin.groesslinger@uni-passau.de; grebhahn@fim.uni-passau.de; kronast@fim.uni-passau.de

S. Apel
University of Passau, Passau, Germany
Saarland University, Saarbrücken, Germany
e-mail: apel@cs.uni-saarland.de

M. Bolten · L. Claus
University of Wuppertal, Wuppertal, Germany
e-mail: bolten@math.uni-wuppertal.de; claus@math.uni-wuppertal.de

S. Chiba
The University of Tokyo, Tokyo, Japan
e-mail: chiba@is.titech.ac.jp

U. Rüde · J. Teich · F. Hannig · H. Köstler · S. Groth · S. Kuckuk · C. Schmitt · J. Schmitt
University of Erlangen-Nuremberg, Erlangen, Germany
e-mail: ulrich.ruede@fau.de; juergen.teich@fau.de; frank.hannig@fau.de; harald.koestler@fau.de; stefan.groth@fau.de; sebastian.kuckuk@fau.de; christian.j.schmitt@fau.de; jonas.schmitt@fau.de

H. Rittich
Forschungszentrum Jülich, Jülich, Germany
e-mail: h.rittich@fz-juelich.de

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_14

www.dbooks.org

behind it and several case studies that demonstrate its feasibility and versatility: high-performance stencil codes can be engineered, ported and optimized more easily and effectively.

1 Overview of ExaStencils

1.1 Project Vision

ExaStencils[1] takes a revolutionary, rather than evolutionary, approach to software engineering for high performance. It seeks to provide a proof of concept that programming can be simplified considerably and that program optimization can be made much more effective by concentrating on a limited application domain. In the case of ExaStencils, it is a subdomain of geometric multigrid algorithms [84]. The idea is to write a program in a *domain-specific programming language* (DSL). In our case, it is an *external* DSL (i.e., a DSL built from scratch rather than embedded in an existing language), called ExaSlang [72]. ExaSlang is stratified into four layers of abstraction. Each layer provides a different view of the problem solution and can be enriched with information to allow for particular optimizations at that layer. ExaSlang's most abstract layer specifies the problem as a set of partial differential equations (PDEs) defined on a continuous domain. The most concrete layer allows the user to specify low-level details for an efficient implementation on the execution platform at hand. Ideally, the domain expert should only be dealing with the first layer (plus some menu-driven options). The ExaStencils code generator should be able to generate all lower code layers while applying a set of optimizations autonomously before producing efficient target code.

1.2 Project Results

This subsection summarizes the challenges that drove the development of ExaStencils and how far we got in the period of SPPEXA funding.

We begin with the delineation of the domain and the mathematical challenges, distinctly from the computer science challenges, that ExaStencils addressed (see Sect. 2). We restricted our attention to the development of smoothers for geometric multigrid methods and the required analysis tools. In this domain, *local Fourier analysis* (LFA) is the method of choice to analyze the developed smoothers and the entire multigrid method. To make LFA useful for our purpose, we extended the method to periodic stencils, covering block smoothers and varying coefficients [10].

[1] www.exastencils.org.

The first major computer science challenge was to cover, with one single source program, a wider range of multigrid solvers than is possible with contemporary implementations based on general-purpose host languages such as Fortran or C++ (see Sect. 3). To this end, we decided not to build ExaSlang on an existing, general-purpose host language but to make it an external DSL. We were able to demonstrate its flexibility already early on in the project by providing a common ExaSlang program for Jacobi, Gauss-Seidel, and red-black solvers for finite differences and finite volumes with constant and linear interpolation and with restriction [53]. Later on, we introduced a way to specify data layout transformations simply by a linear expression—a feature that can aid the development process significantly [46] (see Sects. 3.1–3.2). A smaller task was to decide how to describe aspects of the execution platform in a platform-description language (TPDL) [75] (see Sect. 3.3).

The second major computer science challenge was to reach high performance with our approach to code generation on a wide range of architectures (see Sect. 4). One important aspect here is what information to provide at which layer (see Sect. 4.1). While the syntax of ExaSlang is partly inspired by Scala, Matlab and LaTeX, our target language is C++ with additional features that depend on the execution platform (see Sect. 4.2). We demonstrated that, with the help of our optimization techniques (see Sects. 4.3–4.5), weak scaling could be achieved on traditional cluster architectures such as the *JUQUEEN* supercomputer at the Jülich Supercomputing Center (JSC) [48, 72]. We also achieved weak scaling on the GPU cluster *Piz Daint* at the Swiss National Supercomputing Centre (CSCS). Furthermore, we demonstrated the automatic generation of solvers that can be executed on emerging architectures such as ARM [51] and FPGA platforms [73, 77].

One new concept that ExaStencils introduced into high-performance computing is that of *feature-based domain-specific optimization* [5] (see Sect. 4.6). The central idea is to view a source code, such as a stencil implementation or an application, as a member of a *program family* or *software product line* rather than as an isolated individual, and to describe the source code by its commonalities and variabilities with respect to the other family members in terms of features. A *feature* represents a concept of the domain (e.g., a type of smoother or grid) that may be selected and combined with others on demand. With this approach, a large search space of configuration choices can be reviewed automatically at the level of domain concepts and the most performant choices for the application and execution platform at hand can be identified. To this end, we devised a framework of sampling and machine learning approaches [36, 39, 80] that allow us to derive a *performance model* of a given code that is *parameterized in terms of its features*. This way, we can express performance behavior in terms of concepts of the domain and automatically determine optimal configurations that are tailored to the problem at hand, which we have demonstrated in the domain of stencil codes [29, 30, 32, 36, 39, 59, 80] and beyond (e.g., databases, video encoders, compilers, and compression tools). Our framework integrates well with the other parts of ExaStencils that use and gather domain and configuration knowledge in different phases.

Project ExaStencils came with several case studies whose breadth was to demonstrate the practicality and flexibility of the approach (see Sect. 5). The case

www.dbooks.org

studies are a central deliverable of ExaStencils. They include studies close to real-world problems: the simulation of non-Newtonian and non-isothermal fluids (see Sect. 5.3) and a molecular dynamics simulation (see Sect. 5.4).

We conducted two additional studies at the fringes of ExaStencils, exploring alternative approaches (see Sect. 6). In one, the option of an internal rather than external DSL was explored: ExaSlang 4 was embedded in the mutual host languages Java and Ruby to study the trade-off between the effort of the language implementation and the performance gain of the target code [17]. The outcome was that an embedding is possible but, as expected, with a loss of performance (see Sect. 6.1). In the second study, we implemented a simple multigrid solver in SPIRAL [11] (see Sect. 6.2). The success of the SPIRAL project [24, 61] a decade ago was a strong motivator for project ExaStencils. SPIRAL can handle simple algebraic multigrid solvers but would have to be extended for more complex ones.

1.3 Project Peripherals

Attempts of abstraction and automation in programming have received increased attention in the past two decades in the area of software engineering. High-performance computing has been comparatively conservative in going down this road. The reason is that the demands on performance are much higher than in general software engineering, and the architectures used to achieve it are more complex, notably with large numbers of loosely coupled processors.

The potential of an effective automation grows as the application domain shrinks. Promising domains are much smaller than those of any general-purpose programming language. The extreme is the compiler FFTW [25] that targets a single numerical problem, the fast Fourier transform. As just mentioned, SPIRAL widened the domain to linear transforms (and, lately, beyond [81]). By now, quite a number of optimizing code generators have been proposed that target stencil codes. Patus [18] has a strong focus on autotuning. The strong point of Pochoir [82] is the cache obliviousness of its target code. Pochoir addresses constant stencils and is based on the C dialect Intel Cilk with limited portability to other platforms. Devito [52] performs symbolic computation via SymPy and targets shared-memory architectures, just like Snowflake [90]. Firedrake [63] is primarily for finite-element methods and has adopted multigrid lately. One recent component of it is Coffee [54], which addresses the local assembly in finite-element methods. STELLA [34] and its spiritual successor GridTools generate code for stencil computations on distributed structured grids for climate and weather modeling.

For the domain of image processing, many approaches based on code generation exist that are conceptually similar to the idea behind project ExaStencils. Notable projects include Halide [62], HIPA^{cc} [57], PolyMage [58], and Mint [85]. The specification of image filter kernels is related to the concept of stencils, and many of the basic parallelization techniques are comparable. However, ExaSlang is fundamentally based on computational domains that are multidimensional, as

opposed to the usually two-dimensional data structures used to represent images. Image processing DSLs usually target shared-memory parallel systems, i.e., single compute nodes such as multi-core CPUs or a single GPU. On the other hand, ExaStencils aims at the domain of HPC and, consequently, supports distributed-memory systems and respective parallelization techniques.

2 The Domain of Multigrid Stencil Codes

2.1 Multigrid

The goal of the ExaStencils project has been the automatic generation of efficient stencil codes for geometric multigrid solvers [84] on (semi-)structured grids. Multigrid methods form a class of iterative solvers for large and sparse linear systems. These methods have originally been developed in the context of the numerical solution of partial differential equations, which often involves the solution of such linear systems.

Multigrid methods combine two complementary processes: a *smoothing process* and a *coarse-grid correction*. While each of the two processes is by itself not sufficient to solve the problem efficiently, their combination yields a fast solver.

The smoothing process is usually a straightforward iterative method that converges rather slowly when used on its own. Combining such a process with a coarse-grid correction accelerates the convergence of the resulting method by augmenting the iteration with information obtained on a coarser grid. Since the grid determines the resolution at which the solution is being computed, a multigrid method considers the problem at different resolutions.

A multigrid method performs stencil computations on a hierarchy of fine to successively coarser grids. The overall cost of the method can be reduced further by applying this idea recursively, i.e., instead of two grids, we consider a hierarchy of successively coarser grids. The recursion follows a so-called *cycling strategy*, e.g., a V-cycle or a W-cycle (see Fig. 1). The cycling strategy determines how much work is performed at what level of the grid hierarchy, which also has an effect on the convergence rate of the method. On the coarser grids, less processing power is required.

In summary, to construct a multigrid method, one must choose a set of components: a smoother, an interpolation, a restriction, a coarse-grid approximation, and a cycling strategy. The choice of components influences the number of iterations required to obtain an adequate solution, the computational cost per iteration, and the communication pattern of the method. Furthermore, the behavior of the method depends on the linear system to be solved.

www.dbooks.org

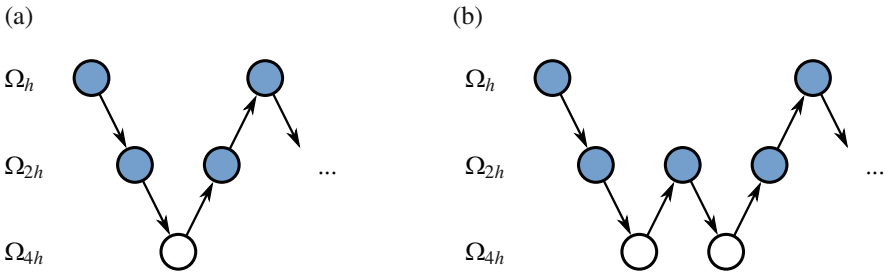


Fig. 1 Cycling strategies across the grid hierarchy. The hierarchy levels are denoted by Ω_h, Ω_{2h}, and Ω_{4h}, progressing from the finest to the coarsest grid. Light circles solve the coarse-grid system directly, and dark circles solve the coarse-grid system recursively. Down arrows symbolize restrictions, up arrows interpolations. (**a**) V-cycle. (**b**) W-cycle

2.2 Advancing the Mathematical Analysis of Multigrid Methods

At the start of project ExaStencils stood the search for a way to estimate the number of iterations that a multigrid method requires to produce an adequate solution. The choice of multigrid components determines the operations that must be performed per iteration. Hence, to estimate the time required per iteration, one just needs to place these operations in an appropriate order and estimate the duration time of their execution. While this is by no means a trivial endeavor, estimating the number of iterations needed is inherently different. In this section, we deal with the latter problem.

To estimate the number of iterations, we must consider the mathematical properties of a given multigrid method for a given set of components and problem. For this purpose, we must determine the contraction properties of the *iteration operator* of the multigrid method for a given configuration. To this end, *local Fourier analysis* (LFA) tells us whether the repeated application of the iteration operator lets the error converge to zero. In particular, we are interested in the resulting rate of convergence.

For project ExaStencils, we had to extend the capabilities of LFA. We wanted to analyze block smoothers, which have a higher arithmetic density than standard smoothers (see Sect. 2.3), and complex problems for which no feasible way of applying LFA was known. Furthermore, we aimed at an automation of the analysis.

At the beginning of project ExaStencils, we considered using and expanding an existing LFA software [88]. However, in the course of the project, it became quickly evident that we needed a more general approach, something beyond a collection of templates that allow to fill in some blank spots but do not facilitate the reuse or recombination of components. Since we intended to explore many possible combinations of various multigrid components, it was not feasible to program an individual LFA by hand for every combination. We developed the *LFA Lab* [65],

which is based on the principle of combining a set of primitive expressions to complicated ones, enabling a much more flexible analysis.

A multigrid method, as well as its components, can be characterized by their iteration operators. An *iteration operator* describes the propagation of the approximation error during the execution of the method. If we denote the error after the k-th iteration with $e^{(k)}$ and the iteration matrix of the method with E, we have the following equation:

$$e^{(j+1)} = Ee^{(j)}$$

The spectral radius $\rho(E)$ and the operator norm $\|E\|$ of E characterize the asymptotic and worst-case error reduction factors of the method, respectively. LFA determines these quantities in a simplified setting.

Let h, with $0 < h \in \mathbb{R}$, be the step size of the infinite d-dimensional grid $G_h := h \cdot \mathbb{Z}^d$. We consider stencil operators on the space of bounded grid functions $\ell_2(G_h) := \{u : G_h \to \mathbb{C} \mid \sum_{x \in G_h} |u|^2 < \infty\}$. A *stencil operator* $A : \ell_2(G_h) \to \ell_2(G_h)$ is a linear operator given by a family $\{s_k\}_{k \in \mathbb{Z}^d}$, $s_k \in \mathbb{R}$, such that

$$Au\,(x) := \sum_{k \in \mathbb{Z}^d} s_k u(x + hk) \quad \text{for} \quad x \in G_h \text{ and } u \in \ell_2(G_h)\,.$$

Stencil operators have a particularly simple form when transformed via the discrete-time Fourier transform.

The *discrete-time Fourier transform* (DTFT) $\mathcal{F}$ is a linear isometry that maps the space $\ell_2(G_h)$ onto $L_2(\Theta_h) := \{u : \Theta_h \to \mathbb{C} \mid \int_{\Theta_h} |u(x)|^2 \,\mathrm{d}x < \infty\}$, where $\Theta_h := [0, \frac{2\pi}{h})^d$. In other words, it represents a function on an infinite grid by a function on a continuous and bounded interval. A stencil operator A in Fourier space, i.e., the operator $\mathcal{F}A\mathcal{F}^{-1}$, is just the multiplication by a function $\hat{a} : L_2(\Theta_h) \to \mathbb{C}$. We call the function $\hat{a}$ the *symbol of* A. Thus, the symbol of a stencil operator is a function that encodes the entire behavior of the infinite-dimensional operator.

The symbol $\hat{a}$ of a stencil operator reveals the desired information about the operator A. We have that

$$\rho(A) = \text{ess-sup}_{\theta \in \Theta_h} |\hat{a}(\theta)| \quad \text{and} \quad \|A\| = \text{ess-sup}_{\theta \in \Theta_h} |\hat{a}(\theta)|\,,$$

where ess-sup denotes the essential supremum. These formulas mean that, to compute the spectral radius or the operator norm of a stencil operator, we must just compute the largest value of the absolute modulus of its symbol $\hat{a}$. Note that, in the definition of stencil operators, we have assumed that stencil s does not depend on position x. However, it can be useful to consider operators whose stencil is allowed to change with the position.

A stencil that depends arbitrarily on the position has no particularly simple form in the Fourier domain. However, we were able to show that periodic stencils do have

www.dbooks.org

a simple representation [10, 66]. A *periodic stencil* depends on the position, but has the same entries repeated periodically across the entire domain. While this does not represent stencils accurately that are variable in the entire domain, at least some variability is reflected in the analysis. We showed that a periodic stencil operator is described, after a proper reordering of the frequency domain, by a matrix of ordinary symbols—more precisely, by matrix symbols from the space $L_2^{n\times m}(\theta_{h'})$ for some appropriate positive h'. Furthermore, we were able to show that there is a one-to-one relationship between matrix symbols and periodic stencils.

Using matrix symbols, similar results for the spectral radius and operator norm hold. For an operator given by a periodic stencil, we have that

$$\rho(A) = \text{ess-sup}_{\theta\in\Theta_{h'}} \|\hat{a}(\theta)\| \quad \text{and} \quad \|A\| = \text{ess-sup}_{\theta\in\Theta_{h'}} \rho(\hat{a}(\theta)) \,.$$

Thus, to obtain the norm and spectral radius of the operator, we must find the largest value of the norm and spectral radius of the matrix $\hat{a}(\theta)$.

The framework of periodic stencils and matrix symbols allows for more advanced problems to be analyzed. It also lends itself to automation via software. Operators that have a matrix symbol can be combined in many ways such that the combination also has a matrix symbol. Thus, we can create a flexible LFA software by using the idea of providing first a set of primitive expressions and then means of combination and abstraction [1].

For example, the iteration operator of the weighted Jacobi method is

$$E_{\text{J}} = I - D^{-1}A \,,$$

where A is the system matrix, D the diagonal part of A, I the identity matrix and $\omega \in \mathbb{R}$ a weighting factor. If we assume that the behavior of A can be modelled sufficiently accurately by a (periodic) stencil operator on an infinite grid, we can replace each matrix by the infinite-dimensional operator given by the corresponding (periodic) stencils to simplify the analysis. Listing 1 shows the computation of the spectral radius of the iteration operator of the Jacobi method for the stencil resulting from the discretization of the Poisson equation using our software LFA Lab [65].

LFA Lab can be used as a simple Python [86] library, but it is more or less an embedded DSL for LFA. The user provides the (periodic) stencils, which yield the corresponding operators. Then, the user combines these operators with interpolation and restriction to an expression describing the desired iteration operator.

```
1 from lfa_lab import *
2 g = Grid(2, [1.0/32, 1.0/32])
3 A = gallery.poisson_2d(g)
4 I = operator.identity(g)
5 omega = 0.8
6 E = I - omega * A.diag().inverse() * A
7 print((E.symbol().spectral_radius()))
```

Listing 1 Implementation of an LFA of the weighted ($\omega = 0.8$) Jacobi smoother for the solution of the Poisson equation using LFA Lab

The code in Listing 1 is essentially a direct implementation of the formula for the iteration operator. However, keep in mind that this formula describes actually a combination of infinite-dimensional operators. When calling the `symbol` method, the software determines automatically a way to represent the given iteration operator via its Fourier matrix symbol, which is a non-trivial procedure.

To compute the matrix symbol of an expression given by the user, appropriate sampling parameters must be determined. For this purpose, LFA Lab has two stages. The first stage extracts and analyzes the expression tree of the formula that the user entered. The second stage then samples the matrix symbol to obtain the spectral radius and operator norm. The power of the software lies in the fact that arbitrarily complex expressions can be analyzed.

A more complex example is the analysis of the two-grid method. It has the iteration operator

$$E_{\mathrm{TG}} = S\,(I - P A_{\mathrm{c}}^{-1} R A)\,S\,,$$

where A_{c} is the coarse-grid operator, P the interpolation, R the restriction, and S the iteration operator of the smoother. Assume that we already have an analysis for the smoother. If we can express P, R, and A_{c} using Fourier matrix symbols, we can combine these with the analysis of the smoother we already have to analyze the two-grid method in its entirety.

In summary, we have constructed a powerful and flexible LFA software. The flexibility comes from a small set of primitive expressions and means of combination and abstraction. The periodic stencil operators are the primitive expressions, mathematical operations are a means of combination, and the Python programming language provides the opportunity of abstraction. The software is used to estimate the convergence rate of a multigrid method for a given set of components and a given problem. The estimate comes as a number of iterations a multigrid method needs to achieve adequate accuracy.

2.3 Advancing Multigrid Components

The choice of the smoothing component in a multigrid method is not always straightforward. Some problems require advanced smoothers. This can be easily appreciated when considering that, since the system contains a zero block, a pointwise relaxation is not possible [84]. The steady-state Stokes equations can be written as follows:

$$\begin{aligned} -\Delta \mathbf{u} + \nabla p &= \mathbf{f}, \quad \text{in} \quad \Omega \\ \nabla \cdot \mathbf{u} &= 0, \quad \text{in} \quad \Omega \end{aligned}$$

www.dbooks.org

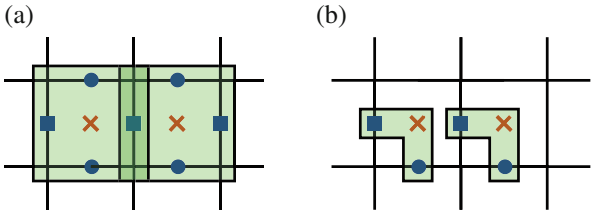


Fig. 2 Unknowns that are included in the blocks of the smoothing steps. (**a**) Vanka smoother. (**b**) Triad smoother

for a given domain Ω with boundary $\partial\Omega$. Here, $\mathbf{u}$ is the vector-valued fluid velocity, p is the pressure, and $\mathbf{f}$ describes an external force.

This linear system of PDEs can be discretized on staggered grids or by using appropriate finite elements. Here, we consider staggered grids in two dimensions. For the Stokes equations, the efficient Vanka smoother has a relatively high computational cost and unsatisfactory parallelization properties due to a process of overlapping block-smoothing steps (see Fig. 2). This made us consider the Triad smoother as an alternative: it provides low computational cost in combination with good parallelization properties [19].

Both block smoothers are based on a collective updating process of unknowns inside one block [87]. As depicted in Fig. 2, the Vanka blocks consist of five unknowns including one pressure and two velocity components in each direction while the Triad smoother comprises the simultaneous update of three unknowns including one unknown of each kind.

Numerical results for periodic boundary conditions in combination with parallelization properties and computational work show the potential of the Triad relaxation method. However, numerical results for the Stokes system with Dirichlet boundary conditions show that the Triad method in its original form has one issue: the convergence rate deteriorates tremendously. To illustrate this, we applied the method to the Stokes equation discretized on a staggered grid in the unit square, with periodic and with Dirichlet boundary conditions. As right-hand side, we chose in both cases $f_{u_x}(x, y) = 2\pi^2 \sin(\pi x)\sin(\pi y) + \pi \cos(\pi x)$, $f_{u_y}(x, y) = 2\pi^2 \cos(\pi x)\cos(\pi y) - \pi \sin(\pi y)$, and $f_p \equiv 0$; the initial guess was zero. Figure 3 shows the different convergence behaviors.

The convergence properties of the Triad smoother can be improved with the following idea: repeat the relaxation process four times while changing the unknowns contained in one block after each iteration, i.e., rotating the "L"-shaped pattern that describes the block to be relaxed. This algorithm, illustrated in Fig. 4, improves the convergence significantly. When applying the four iterations of the proposed smoother, it is very similar to one iteration of Vanka. Thus, a smoother that has not been considered as an option before becomes a viable alternative to established smoothers for systems of PDEs. In addition, the order in which the boxes are updated can be varied. For more details and further results, see the dissertation of Lisa Claus [19].

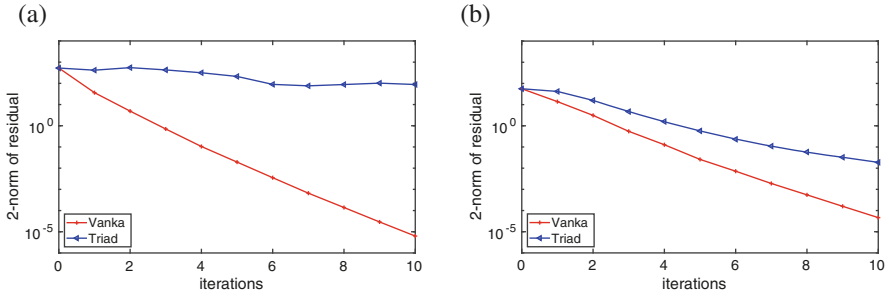


Fig. 3 Convergence behaviors of the two block smoothers applied to the Stokes equations discretized using staggered grids in the unit square with periodic and Dirichlet boundary conditions. (**a**) Dirichlet boundary conditions. (**b**) Periodic boundary conditions

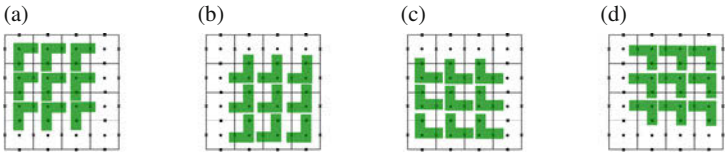


Fig. 4 Order of iterations of the advanced block smoother. (**a**) Step 1. (**b**) Step 2. (**c**) Step 3. (**d**) Step 4

3 Stencil-Specific Programming in ExaStencils

The central element in the ExaStencils approach is its stencil-specific programming language ExaSlang and its code generator. Then there is also a language for specifying properties of the execution platform. The two languages are discussed in this section. The code generator is the subject of the following section.

3.1 The Domain-Specific Language ExaSlang

The idea of ExaStencils is to support the domain-specific programming and optimization of stencil codes by providing different layers of abstraction, specifying the various aspects of the stencil code at the respectively suitable layer, and exploiting domain information available at that layer for an optimization of the specification. ExaSlang comes in four layers: from ExaSlang 1, the most abstract, to ExaSlang 4, the most concrete (see Fig. 5).

- *Layer 1: the continuous problem*
  This is the layer for the scientist or engineer who needs the solution of the PDE. The problem is specified as a continuous equation. The present implementation supports Unicode and LaTeX symbols. Optional inputs are the specification of

www.dbooks.org

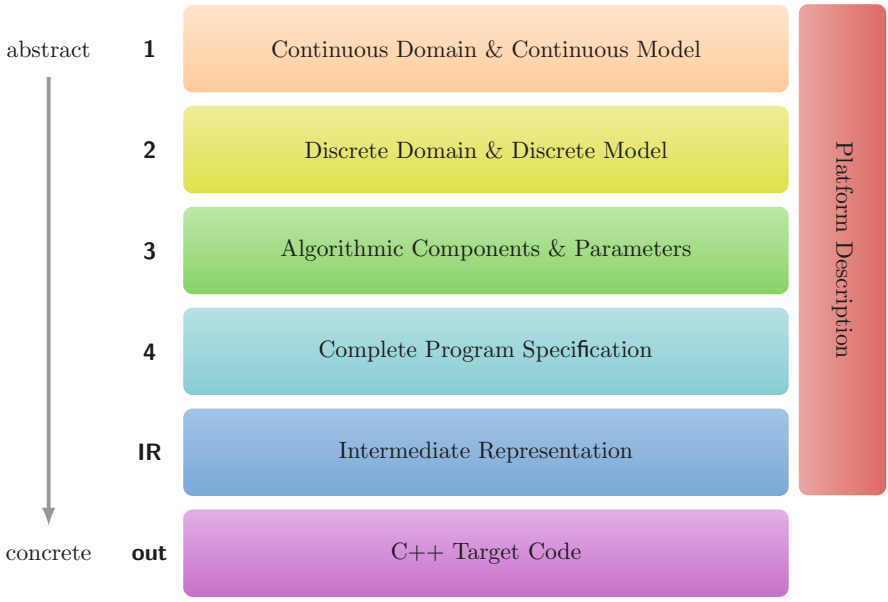


Fig. 5 ExaSlang layers of abstraction

discretization and solver options used to autogenerate lower layers. There is also support for an automatic finite-difference discretization of operators.

- *Layer 2: the discrete problem*
  This is the most abstract layer that provides an executable description. Discretized functions are fields (data type, grid location), tied to a computational domain. Geometric information is provided in the form of *virtual fields* resolved to constants or field accesses. Discretized operators are provided as stencils or stencil templates.
- *Layer 3: the solver*
  At this layer, multigrid appears in the form of the specification of a solver for the discrete problem, either provided by hand or set up automatically. The implementation supports a Matlab-like syntax.
- *Layer 4: the application*
  This layer of ExaSlang can describe a full application, including communication, input/output, evaluation and visualization, but is still more abstract than C++ or Java in that it contains language constructs specific to multigrid. Optimizations at this layer include the tuning of communication patterns and data structure layouts.

Yet more concrete, but not accessible to the user, is an intermediate representation (IR) in which most code refinements take place and which forms the base for the generating of the target C++ code.

One may write one's program at the ExaSlang layer of one's choice and let the code generator refine it to a more concrete layer. One may also modify generated code to implement certain aspects that cannot be expressed at the chosen programming layer. However, the IR representation is not meant to be modified by the user.

Through its four layers, ExaSlang evolves from a declarative language at layer 1 to an imperative programming language at layer 4. Besides the standard data types that represent floating-point and integer numbers or strings, domain-specific data types represent vectors and matrices to be used for coupled systems of equations. Assembly of global vectors and matrices is not supported since we focus on local computations using stencils. In contrast, stencils are declared globally in ExaSlang programs. Furthermore, fields—corresponding to vectors that may represent the PDE's right-hand side or an approximation of the unknown function—are declared globally as well. Via declaration, certain settings important to parallelization may be specified by the user. One example is the size of ghost layers (also called halo layers), as depicted in Listing 5 in Sect. 3.2.

3.2 An ExaSlang Example

To illustrate how our language stack can be used to implement different aspects of partial differential equations (PDEs) solvers, let us take the Poisson equation

$$\begin{aligned} -\Delta u &= f \quad \text{in} \quad \Omega\,, \\ u &= g \quad \text{on} \quad \partial\Omega \end{aligned}$$

for a given domain Ω—here the unit square—Dirichlet boundary conditions $g = cos(\pi x) - sin(2\pi y)$ and the right-hand side $f = \pi^2 cos(\pi x) - 4\pi^2 sin(2\pi y)$. Listing 2 shows an exemplary layer 1 code for this specification. From it, our generator is able to derive representations at subsequent layers. The refinement methods employed in this process are described in Sect. 4.1 further below. Listings 3 and 4 illustrate variants similar to the autogenerated ones expressing the discretized version of the given equation (Listing 3) and the multigrid algorithm used to solve for it (Listing 4). Based on them, a complete layer 4 program can be assembled, comparable to the one illustrated in Listing 2. In the example code at the lower layers, parts of the source code have been omitted for the sake of compactness. A complete specification and examples of other PDEs are part of Sebastian Kuckuk's dissertation [47].

www.dbooks.org

```
Ω = ( 0, 1 ) × ( 0, 1 )

u ∈   Ω = 0.0
u ∈ ∂ Ω = cos ( π x ) - sin ( 2 π y )

f ∈   Ω = π^2 cos ( π x ) - 4 π^2 sin ( 2 π y )

op = - Δ

uEq: op * u == f

/* discretization and solver hints (see Subsection 4.1) */
```

Listing 2 ExaSlang 1 code for the complete specification of the 2D Poisson problem

```
global from [ 0, 0 ] to [ 1, 1 ]

Solution with Real on Node of global = 0.0
Solution@finest on boundary =
  cos ( PI * x ) - sin ( 2.0 * PI * y )
Solution@(all but finest) on boundary = 0.0

RHS with Real on Node of global =
  PI**2 * cos ( PI * vf_nodePos_x ) -
  4.0 * PI**2 * sin ( 2.0 * PI * vf_nodePos_y )

Laplace from Stencil {
  [ 0,  0] =>  2.0 / ( vf_gridWidth_x**2 ) +
               2.0 / ( vf_gridWidth_y**2 )
  [-1,  0] => -1.0 / ( vf_gridWidth_x**2 )
  [ 1,  0] => -1.0 / ( vf_gridWidth_x**2 )
  [ 0, -1] => -1.0 / ( vf_gridWidth_y**2 )
  [ 0,  1] => -1.0 / ( vf_gridWidth_y**2 )
}

SolEq {
  Laplace * Solution == RHS
}
```

Listing 3 ExaSlang 2 code for a complete specification of the 2D Poisson problem

```
Field Residual from Solution
override bc for Residual with 0.0

Operator Restriction  from default restriction
  on Node with 'linear'
Operator Prolongation from default prolongation
  on Node with 'linear'

Function Smoother@all {
  repeat 3 times {
```

```
    Solution += diag_inv ( Laplace ) * ( RHS -
      Laplace * Solution ) where (i0 + i1) % 2 == 0
    Solution += diag_inv ( Laplace ) * ( RHS -
      Laplace * Solution ) where (i0 + i1) % 2 == 1
  }
}

Function VCycle@coarsest {
  /* implementation of a coarse-grid solver */
}

Function VCycle@(coarsest + 1 to finest) {
  Smoother ( )

  Residual = RHS - Laplace * Solution
  RHS@coarser = Restriction * Residual

  Solution@coarser = 0.0
  VCycle@coarser ( )

  Solution += Prolongation@coarser * Solution@coarser

  Smoother ( )
}
```

Listing 4 ExaSlang 3 implementation of a V(3, 3)-cycle using an RBGS smoother

```
Layout DefLayout<Real, Node >@all {
  duplicateLayers = [1, 1] with communication
  ghostLayers     = [1, 1] with communication
}

Field Solution< global, DefLayout, /* bc's */ >@finest
Field Solution< global, DefLayout, 0.0 >@(all but finest)
Field RHS     < global, DefLayout, None >
Field Residual< global, DefLayout, 0.0 >

/* operators as on layers 2 and 3 */

Function Smoother@all {
  color with (i0 + i1) % 2 {
    loop over Solution {
      Solution += omega * diag_inv ( Laplace ) *
                  ( RHS - Laplace * Solution )
    }
    communicate Solution
  }
}

/* VCycle functions */

Function Application {
  /* initialization */
```

www.dbooks.org

```
27
28  repeat 10 times {
29    VCycle@finest ( )
30  }
31
32  /* de-initialization */
33}
```

Listing 5 ExaSlang 4 code of a full application with a fixed number of V-cycles to solve for Poisson's equation discretized with finite differences

3.3 The Target-Platform Description Language

To be able to optimize a code adequately, one must know details of the execution platform. Our code generation process is governed by more than one hundred parameters that allow to select specific code refinements or to set device-specific properties. Examples include the use of vector units on CPUs and the corresponding instruction set to use, e.g., SSE or AVX on $\times 86$ CPUs, NEON on ARM-based CPUs or even QPX for IBM's POWER architecture. This yields a design space that is too large for users to be able to specify a (near-)optimal configuration of code generation settings. However, we may be able to derive sensible parameters from a structured description of the target platform. To this end, one element of ExaSlang is the so-called target platform description language (TPDL) [75]. One design goal was to increase the modularity and reusability of hardware component descriptions, such as CPUs or accelerators, to let users compose systems based on a repository of ready-made parametric snippets. By treating these in a fashion similar to the class concept in object-oriented programming languages, users can infer instantiations with parameters set appropriately. A short example describing an accelerator card is provided in Listing 6. It enumerates a number of technical hardware details, but also contains the important software information on the compute capability, i.e., which features of the target technology CUDA may be used.

```
<gpu name="Tesla_V100" role="worker">
  <param name="compute_capability" value="7.0" />
  <param name="api" value="cuda" />
    <memory size="16" unit="GigaByte" Type="HBM2">
      <param name="bandwidth" value="900" unit="GigaBps"/>
    </memory>
  <core quantity="5120" frequency="1246"
  frequency_unit="MegaHz" />
</gpu>
```

Listing 6 ExaSlang description of an accelerator card

```
val workers = predefinedQuery(tpdlTree, GetWorkingUnits)
val workers50 = workers.filter(w => w.power > Watt(50))
workers50.foreach(System.out.println(_))
```

Listing 7 Query to enumerate working units consuming more than 50 W of power

A small, yet flexible library supports information retrieval from a TPDL specification. This enables DSL developers to check for certain information and aggregate or evaluate characteristics of the target platform. Developers need not worry about the instantiations and their parameters but may just use the discrete specification, since all the processing required has already been done. For many recurring tasks, predefined queries are available. Listing 7 shows a predefined query to return all working units in a system. Its result is filtered to retain only working units that consume more than 50 W of electrical power.

4 The ExaSlang Code Generator

4.1 Refinement of ExaSlang Programs

The overall goal of ExaStencils has been to enable users to choose the layer most appropriate for them and code exclusively at this layer, e.g., by providing a continuous formulation of the problem to be solved at layer 1 and nothing more. Ideally, our framework would then automatically derive suitable discretizations (layer 2), solver components (layer 3) and parallelism particulars (layer 4). However, in practice, this requires *domain knowledge* whose automatic inference is beyond the capabilities of present software technology. We address this issue by introducing *hint* specifications which allow us to progress automatically to subsequent layers of ExaSlang. For example, at layer 1, *discretization hints* may be supplied. As Listing 8 shows, continuous functions are discretized at certain points of a computational grid, such as node positions. We also support the specification of operator discretization using finite differences. An optional renaming is also possible at this stage.

```
DiscretizationHints {
  u  => Solution                          on Node
  op => Laplace  with "FiniteDifferences" on Ω order 2
  uEq
}
```

Listing 8 A discretization hint block for a scalar equation in ExaSlang 1

Then, we combine the equation provided at layer 1 and the discretization hints to synthesize a discretized form of the equation at layer 2. While layer 2 expresses the problem to be solved, its solution is specified at layer 3. This can be achieved either by implementing a suitable iterative solver by hand or by issuing a directive for our *generate solver* interface. Listing 9 illustrates such a directive.

www.dbooks.org

```
generate solver for Solution in SolEq with {
  solver_smoother_numPre          = 3
  solver_smoother_numPost         = 3
  solver_smoother_coloring        = "red-black"
  solver_cgs                      = "ConjugateGradient"
}
```

Listing 9 A generate solver statement in ExaSlang 3

For cases in which no further modification is required, matching *solver hints* may be provided at the upper layers to set up the code in Listing 9 automatically, allowing users to work exclusively at one layer. The generated solver is by default a geometric multigrid variant. In the concrete case of Listing 9, our framework would generate a standard V-cycle using three pre- and post-smoothing steps of a red-black Gauss-Seidel (RBGS) and a conjugate gradient (CG) coarse-grid solver. Frequently, minor adaptations of the generated solver are necessary. They can either be implemented by taking the implementation generated at layer 3, adapting it and replacing the original generate solver directive with the result. A more generic approach is to add *modifiers* to the generate solver directive. They target usually a certain *stage* of the multigrid solver, e.g., the restriction or the correction, at one or more levels of the hierarchy. These stages can either be replaced completely with custom layer 3 code, or arbitrary layer 3 statements may be added to be executed before or after the stage. A more complex option exists for *smoother stages*, as illustrated in Listing 10 for the case of a block smoother to be used when solving for the Stokes equations on a staggered grid (see Sect. 5).

```
smootherStage {
  loopBase p solveFor {
    u@[0, 0] u@[1, 0]
    v@[0, 0] v@[0, 1]
    p
  }
}
```

Listing 10 ExaSlang 3 code for a smoother stage for the 2D Stokes problem

After the algorithmic specification at layer 3 is completed, it must be transformed to a program at layer 4. This requires, most importantly, the addition of data layout information for fields, loops to compose kernels, and suitable communication statements. Listings 11 and 12 illustrate the first half of an RBGS smoother.

```
Solution += ( diag_inv ( Laplace )
              * ( RHS - Laplace * Solution )
            ) where (i0 + i1) % 2 == 0
```

Listing 11 ExaSlang 3 code for the first half of an RBGS smoother

```
communicate Solution
loop over Solution where ( i0 + i1 ) % 2 == 0 {
  Solution += diag_inv ( Laplace ) * ( RHS - Laplace *
      Solution )
}
```

Listing 12 ExaSlang 4 code for the first half of an RBGS smoother derived from its ExaSlang 3 counterpart in Listing 11

This description of features is by no means complete. More detail is available in Sebastian Kuckuk's dissertation [47].

We designed the language stack of ExaSlang to enable maximum flexibility for users. They can either work at one level exclusively and make use of the hint system to generate more concrete specifications automatically, or implement different parts of their application at multiple levels, or mix both approaches.

4.2 Generation of Target Code

After the domain-specific program has been refined to a complete program specification (ExaSlang 4), the IR code is subjected to many non-algorithmic transformations (recall Fig. 5 on page 416). Among others, parallelization techniques are applied to the code (see Sect. 4.4), memory layouts are selected and applied to variables (see Sect. 4.3), required code fragments are inserted, and finally the program is written to disk as C++ files.

To express all these steps in a short syntax, we developed the code-transformation framework *Athariac* [76]. It allows to modify the tree-based representation of a program—called the *abstract syntax tree* (AST)—by using simple, yet powerful rewrite rules. Essentially, we specify a pattern in the AST, such as the node that represents a certain ExaSlang statement, and then define a structure with which to replace it. Of course, we can also remove nodes by specifying an empty replacement. A single rewrite rule is called a *transformation*, and a group of transformations is called a *strategy*. In Listing 13, a simple strategy for code-generation time evaluation of mathematical expressions is presented. First, the two identifiers *E* and *PI* are replaced by their constant numerical values. Then, we look for constant numerical values to be summed, e.g., $2.1 + 3.1415$, and replace them with the actual result of the addition. Note that the constant-folding transformation must be applied multiple times for expressions involving multiple additions.

```
var s = new DefaultStrategy("simple strategy")
s += new Transformation("resolve constants", {
  case Identifier("PI") => RealConstant(3.1415)
  case Identifier("E")  => RealConstant(2.7183)
})
```

www.dbooks.org

```
s += new Transformation("constant folding", {
  case Addition(a : RealConstant, b : RealConstant)
    => RealConstant(a + b)
})
s.apply
```

Listing 13 A strategy with two simple transformations

By chaining and conditional execution of strategies in a fixed order, ExaSlang programs can be refined for specific program configurations. Depending on the selected code-generation parameters and input-program size, between 200 and 300 transformations are required to generate C++ code. In Fig. 6, the trajectory of a program from the intermediate representation to C++ output is depicted to illustrate the basic approach that applies to every program. Most of the optimizations are

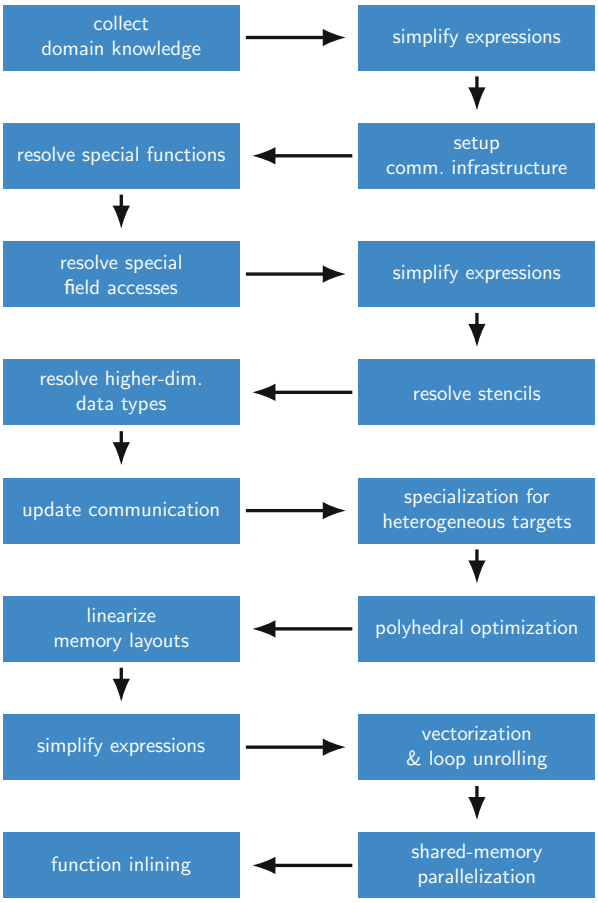


Fig. 6 The transformation trajectory of an ExaSlang program from its IR to C++

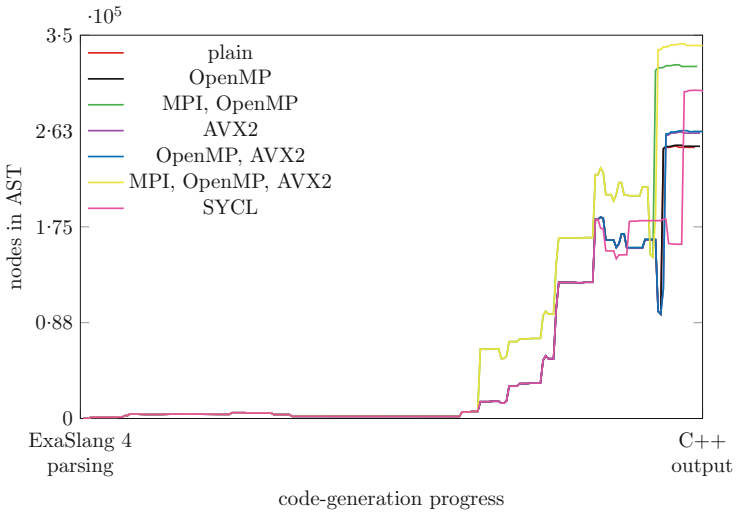


Fig. 7 AST size (number of nodes) during code generation for different variants of the 3D optical flow application, starting at ExaSlang 4

applied in this trajectory. Naturally, a concrete code-generation path may differ, as users can skip certain optimizations, or transformations may not be applied because the preconditions required are not satisfied. For example, it does not make sense to generate code for heterogeneous target devices if none exist.

In Fig. 7, program sizes during the code-refinement process of the three-dimensional optical flow application (see Sect. 5.3) are depicted for a selection of different parallelization techniques. For all variants, we see different stages of the process. First, information on the program is gathered and preparations are done, i.e., setting array sizes and memory layouts. Next, minor code refinements are applied, such as the replacement of stencil convolutions with their corresponding computational rules. Then, depending on the parallelization technique, memory layouts are imposed and loops are generated, respectively modified. The execution times of Athariac's code generator are usually on the same order of magnitude as an invocation of the target C++ compiler, i.e., within a few seconds to minutes. More detail on Athariac can be found in a recent issue of the Proceedings of the IEEE [76] and in Christian Schmitt's dissertation [71].

4.3 Target-Specific Optimization

Stencil codes have a very simple structure but they are difficult to optimize since the memory bandwidth usually acts as a performance brake. In project ExaStencils, a diverse set of optimizations has been developed to overcome this. We describe a

www.dbooks.org

small selection briefly here. More detail on all implemented techniques, including those not presented here, can be found in Stefan Kronawitter's dissertation [43].

Data Layout Transformations Much effort of adapting stencil codes from one application or execution platform to another goes into making the data layout fit for best efficiency. One major limiting factor is usually the available memory bandwidth. Examples of such adaptations are color splitting for multi-color kernels or switching between an array of structs (AoS) and a struct of arrays (SoA).

ExaSlang admits flexible layout transformation directives for the specification of arbitrary affine transformations that are then applied automatically by the code generator [46]. Even though different variants of a color splitting or an SoA-to-AoS transformation may be most commonly used, the ExaStencils code generator is by no means restricted to them. Moreover, the only modification required to adapt the memory layout of any field is the insertion of an affine function that specifies the transformation. No other part of the application, including data initialization and communication, must be modified to make the new layout generally available. Explicit albeit very simple modifications to other parts are only necessary if they are meant to use different data arrangements. In this case, additional fields must be inserted for each layout and copy kernels are necessary.

This approach avoids unnecessary changes in the source code and constitutes a big advance in the ease of testing and evaluating different memory layout schemes in order to identify the best memory layout. There are other systems that offer similar transformation devices but not in this generality.

Polyhedral Code Exploration Besides a layout modification, an affine transformation can lead to the most efficient implementation of a frequent structure in stencil codes: the loop nest. A popular approach to selecting such transformations automatically is the polyhedron model for loop optimization [22]. As for the data layout, the search is also here for best data locality. However, established automatic transformation techniques based on the PLuTo algorithm [6, 12, 13] fail to yield optimal results.

Our first attempt to select better transformations was a specialized variant of the PLuTo algorithm available in the ExaStencils code generator. While it is capable of detecting very good schedules for some stencil codes, it can also encounter problematic ones. For example, it fails completely for RBGS kernels. Thus, we developed a new, optimized, multi-dimensional polyhedral search space exploration for the ExaStencils code generator [44] that obtains in several cases better results than existing approaches, such as different PLuTo variants or PolyMage [58]. It also has the capability of specializing the search for the domain of stencil codes, which reduces the exploration effort dramatically without significantly impairing performance. An extreme but still beneficial approach is to choose the first schedule selected by our specialized search without any further evaluation. This may not lead to the best performance but it avoids the overhead of a complete exploration—and the performance improvement is still satisfactory: in most experiments it was only a few percentage points below the best variant explored.

Vectorization A third optimization focuses on the vector units available in most processor architectures that provide single-instruction-multiple-data (SIMD) parallelism. Their use is typically mandatory for highest performance. However, each architecture comes with its own vector instruction set. Intel ×86 features Streaming SIMD Extensions (SSE) or Advanced Vector Extensions (AVX) in several different versions, IBM's BlueGene/Q provides Quad Processing eXtension (QPX), and some ARM processors implement the Neon instruction set. Even though all of these sets target the same problem, their implementations differ not only in detail but also in key aspects, such as the way in which data can be fetched from main memory.

As a remedy, contemporary compilers are equipped with rudimentary automatic vectorization capabilities most of which are, unfortunately, not very effective. On top, the more advanced compilers can exclude some popular architectures and be costly or not widely available. Since the ExaStencils code generator comes with its own vectorization phase [43, 76], it avoids any dependence on a special target compiler. Currently, it supports Intel SSE3, AVX, and AVX2, as well as IBM's QPX and ARM Neon.

4.4 Parallelization

To parallelize ExaSlang applications automatically, mainly two concepts must be implemented [48]. First, data must be partitioned and distributed across the available compute resources and, second, data between the partitions must be synchronized periodically. We realize the former by splitting our computational domain into *blocks* which are further subdivided into *fragments*. Each fragment holds a part of the computational grid and all data associated with it. This hierarchical approach is depicted in Fig. 8 and permits an efficient mapping to different execution platforms. For instance, blocks can be mapped to MPI ranks while each fragment inside is handled by a distinct OpenMP thread. Mapping single fragments to accelerators is also possible, as explained later on. The synchronization of data can be controlled by users at layer 4 via *communicate* directives. They specify the field to be communicated and can additionally be parameterized to communicate only certain parts, e.g., specific ghost layers. Each communicate directive triggers the generation

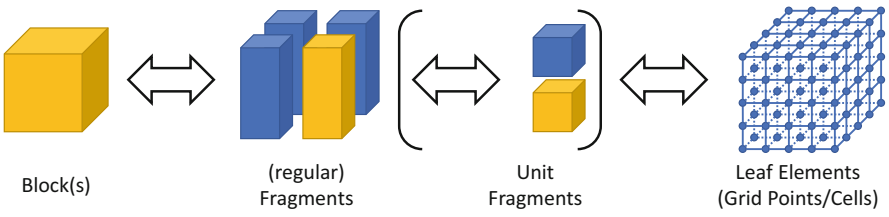


Fig. 8 Partitioning of the computational domain in ExaStencils [48]

www.dbooks.org

of a function that implements the corresponding behavior. This permits the reuse of functions, which can become quite lengthy if the same communication is reissued throughout an application. The function contains code specifying an MPI exchange for fragments residing in different blocks and simple copy kernels for fragments inside the same block. For the interblock case, this includes copying the required data to buffers, calling the appropriate MPI functions, waiting for their completion and copying back data received. The communication buffers required are set up automatically by our framework and, additionally, are shared between multiple communication routines where possible. This minimizes the memory footprint as well as the allocation and deallocation overhead.

Kernels updating data in one fragment can additionally be parallelized. This can be done either with OpenMP or, if available, via an accelerator, e.g., a GPU or reconfigurable hardware such as an FPGA.

Accelerators To accelerate computations with Nvidia GPUs, we require multiple steps during code generation and developed a back-end to emit corresponding CUDA[2] code. First, fields must be extended to manage a CPU and a GPU version. We also introduce flags that keep track of field data being changed on the host or device side at this point. Next, compute kernels are transformed to their CUDA counterparts and wrapped by an interface function passing variables and fields, used inside the kernel, as parameters. Then we replace the original kernel with a user-controlled condition branching to either the original CPU kernel or the new GPU kernel. Both variants are extended to set the flags previously described after execution. The same flags can also be used to control the use of added copy operations between host and device. This ensures that data is always synchronized correctly while the overhead is minimized.

As an alternative to CUDA, SYCL is a new technology for the inclusion of the OpenCL ecosystem into the world of C++. It strives to combine the strengths of both worlds, e.g., by allowing to use custom data structures and templates inside computational kernels, or by providing an implementation of the parallel C++ Standard Template Library that can be executed not only on multicore CPUs, but also on accelerators. Furthermore, it aims at a reduction of OpenCL boilerplate code by enabling the direct issuance of `parallel_for` statements instead of forcing the user to declare kernels, map their arguments, etc. Additionally, it detects automatically and initiates memory transfers between host and device.

We had to specialize our code generation workflow slightly to address the characteristics of SYCL. While, in ExaSlang, the dimensionality of a kernel can be arbitrarily large, SYCL only supports up to three-dimensional kernels, i.e., 3D iteration domains. Because of this, we implemented a custom mapping to linearize a kernel with more than three dimensions to a single dimension while maintaining correct indexing. Furthermore, SYCL has currently no built-in support for reduction operators, such as the sum of all data items. Since multigrid methods depend on

[2]CUDA is a proprietary application programming interface and de-facto standard for Nvidia GPUs.

reduction operators for the computation of norms used to approximate the reduction of the residual, we implemented our own reduction operator and used a template function that is called in the corresponding places. Our approach uses standard parallelization techniques that should perform well on CPUs and GPUs.

SYCL separates work items into work groups in the same sense in which these concepts are used by CUDA and OpenCL. While this concept is not paramount, our code generator can optionally specify the granularity of work groups and modify data accesses inside the computational kernels correspondingly.

SYCL is merely a specification, as of yet without a complete and optimized implementation. We verified our SYCL back-end with the commercial product ComputeCpp by Codeplay and also with the experimental open-source implementation triSYCL[3] [33]. Although SYCL still needs to mature, it has great potential as a unified middleware for a multitude of devices: in addition to targeting CPUs and GPUs, the two major vendors of reconfigurable hardware, Xilinx and Intel/Altera, are actively working to accept SYCL code. This will allow us to extend the pool of ExaSlang-supported hardware targets without having to develop a separate code-generator back-end, such as the one presented in the next paragraph.

Reconfigurable Hardware A particularly interesting target platform for scientific computing is reconfigurable hardware, such as in field-programmable gate arrays (FPGAs). They are increasingly used in high-performance computing because of their high throughput and energy efficiency. We started with extending an existing code generator for imaging processing to produce a description of a simple V-cycle for further processing using C/C++-based high-level synthesis via Vivado HLS for Xilinx FPGAs [69, 70]. This laid the base for a custom Vivado HLS back-end in our code generator, resulting in the ability to emit algorithm descriptions for Vivado HLS stemming from user input in ExaSlang [73]. To test the capabilities of the HLS back-end, we considered a starting grid of size 4096×4096 and used a V(2,2)-cycle. For the coarse-grid solution at a size of 32×32, a corresponding number of Jacobi smoother iterations was applied. The biggest difference to the execution on a CPU or GPU is that all multigrid operators were laid out spatially on the FPGA chip. Another essential consideration in mapping algorithms to reconfigurable hardware is the handling of memory and data transfers. FPGAs provide only very scarce amounts of on-chip memory, called block RAM, that can be accessed with full speed. This implies that data needs to be stored off-chip, e.g., in RAM that can be found on the FPGA board or even in the host computer's memory. As a consequence, these transfers need to be supported by the hardware with corresponding buffers, which are mapped to first-in-first-out (FIFO) hardware buffers. In an FPGA, FIFO buffers can be either composed of registers or implemented by on-chip block RAM.

Another important concern of designing reconfigurable hardware descriptions is the usage of computational resources on the chip. For our initial experiments, we used single-precision floating-point data types and were able to fit the complete

[3] https://github.com/triSYCL/triSYCL.

www.dbooks.org

design on a mid-range Kintex 7 FPGA. On this chip, our design was able to outperform an Intel i7-3770 in terms of throughput by a factor of approximately 3. Switching from single to double precision does not change any performance aspects but merely requires more chip resources. As a result, we had to switch to the larger Virtex 7 FPGA to house the design but were able to retain the performance advantage.

Based on these results, we worked on incorporating a more relevant coarse-grid solution algorithm into our hardware designs. By a clever arrangement of the required memory buffers and transfers, we were able to implement a conjugate-gradient solver and map it to the Virtex 7 using double-precision floating-point numbers [77]. This algorithm is usually not well-suited for FPGAs because of its random memory-access patterns that break the preferable pipelining model. As a consequence, data needs to be stored on-chip in block RAM to retain performance. However, this takes away resources required by other multigrid components, e.g., smoothers and inter-grid operators. We conducted several experiments and were able to outperform an Intel Xeon E5-1620 v3 in many cases [77]. As an optimization, we overlapped successive V-cycles, i.e., we were able to reuse multigrid operators on the FPGA when they were no longer required by the previous V-cycle iteration provided the data dependences were respected. Theoretically, this should nearly double the throughput, which was confirmed in our numerical experiments.

4.5 Compositional Optimization

The ExaStencils framework enables the automatic generation of geometric multigrid solvers from a high-level representation either by using the generate solver interface or by means of a direct specification of its algorithmic components at layer 3. However, in many cases, the construction of an efficient and scalable multigrid method is a difficult task requiring extensive knowledge in numerical mathematics. Therefore, it is typically performed manually by a domain expert. Part of ExaStencils' vision is to automate all steps from the specification of a problem in the continuous domain (layer 1) to the generation of an efficient implementation on the target platform. To achieve this goal, we construct geometric multigrid methods for solving systems of linear equations as program optimization tasks. We have developed a context-free grammar for the automatic construction of solver instances from a given set of algorithmic options, whose production rules are shown in Fig. 9 for the case of pointwise smoothers. Each rule defines the list of expressions by which a certain production symbol, denoted $\langle \bullet \rangle$, can be replaced. To generate a multigrid expression, starting with the symbol $\langle S \rangle$, this process is repeated recursively until the produced expression contains only terminal symbols or the empty string λ. A more detailed description, including the semantics of the expressions generated following this grammar, can be found elsewhere [78].

The effectiveness of an iterative method is determined by two objectives: its rate of convergence and its compute performance on the target platform. To estimate

$$
\begin{aligned}
\langle S \rangle &\vDash \text{ITERATE}(\langle c^h \rangle,\ \omega,\ \langle \mathcal{P} \rangle) \\
\langle c^h \rangle &\vDash \text{APPLY}(\langle B_h \rangle,\ \langle c^h \rangle) \ \mid\ \text{RESIDUAL}(A_h,\ \langle s^h \rangle) \\
\langle s^h \rangle &\vDash \text{ITERATE}(\langle c^h \rangle,\ \omega,\ \langle \mathcal{P} \rangle) \\
\langle s^h \rangle &\vDash \text{ITERATE}(\text{COARSE-GRID-CORRECTION}(P_{2h},\ \langle c^{2h} \rangle,\ \omega),\ \omega,\ \lambda) \ \mid\ (u_0^h,\ f_0^h,\ \lambda,\ \lambda) \\
\langle B_h \rangle &\vDash \text{INVERSE}(\text{DIAGONAL}(A_h)) \\
\langle c^{2h} \rangle &\vDash \text{APPLY}(\langle B_{2h} \rangle,\ \langle c^{2h} \rangle) \ \mid\ \text{RESIDUAL}(A_{2h},\ \langle s^{2h} \rangle) \\
\langle c^{2h} \rangle &\vDash \text{COARSE-CYCLE}(A_{2h},\ u_0^{2h},\ \text{APPLY}(R_h,\ \langle c^h \rangle)) \\
\langle s^{2h} \rangle &\vDash \text{ITERATE}(\langle c^{2h} \rangle,\ \omega,\ \langle \mathcal{P} \rangle) \ \mid\ \text{ITERATE}(\text{APPLY}(P_{4h},\ \langle c^{4h} \rangle),\ \omega,\ \lambda) \\
\langle B_{2h} \rangle &\vDash \text{INVERSE}(\text{DIAGONAL}(A_{2h})) \\
\langle c^{4h} \rangle &\vDash \text{APPLY}(A_{4h}^{-1},\ \text{APPLY}(R_{2h},\ \langle c^{2h} \rangle)) \\
\langle \mathcal{P} \rangle &\vDash \text{RED-BLACK PARTITIONING} \ \mid\ \lambda
\end{aligned}
$$

Fig. 9 A formal grammar for the generation of multigrid solvers

both objectives for a given multigrid solver expression in reasonable time, we employ automated local Fourier analysis (LFA), as described in Sect. 2.2, for the first and a roofline performance model [89] for the second objective. Our goal is to obtain the set of Pareto-optimal solvers with respect to both objectives. A direct enumeration and evaluation of all possible configurations leads to an exponential operation increase with the number of multigrid levels. Thus, a brute-force search is infeasible in most cases. As a remedy, we employ evolutionary computation, a family of stochastic optimization methods stemming from the field of artificial intelligence and inspired by the principle of biological evolution. These methods evolve a population of candidate solutions, which are iteratively improved through the use of so-called genetic operators. The operators are specifically designed for the given problem representation, in our case the set of expressions that can be generated according to the grammar described in Fig. 9. After the optimization is completed, we transform the expression of each Pareto-optimal solver to an algorithmic representation which is emitted in the form of ExaSlang 3 code that can then serve to evaluate the solver's rate of convergence and compute performance on the target hardware. As a first step, we have implemented our optimization approach in Python,[4] using the libraries LFA Lab [65] and DEAP [23]. We point the interested reader to preliminary results of the optimization of multigrid solvers for the steady-state heat equation on a multi-core CPU [78].

In the future, we intend to integrate this implementation into the generate solver interface for the automatic construction of efficient and scalable geometric multigrid solvers based on a given specification.

[4] https://github.com/jonas-schmitt/evostencils.

www.dbooks.org

4.6 Feature-Based Domain-Specific Optimization

One new concept that ExaStencils introduced into high-performance computing is that of *feature-based domain-specific optimization* [5]. The central idea is to look at a source program (e.g., a stencil code or application) not as an isolated individual but as a member of a *program family* or *software product line* and to specify it by its commonalities and variabilities with respect to the other family members in terms of features. A *feature* represents a domain concept (e.g., a type of smoother or grid) that may be selected and combined with others on demand. With this approach, a wide search space of configuration choices can be reviewed automatically at the level of domain concepts and the most performant choices for the application and execution platform at hand can be identified.

Features and configuration options can have a significant influence on the performance of the generated code. The influences of some might already be known to the developer of the system, but other influences may be opaque and likewise may be influences that arise from interactions among features, called *feature interactions*. Also, the influences that one may expect based on theory and domain knowledge may not match the actual influences in the program, which often depend on implementation details.

Ideally, a developer or user knows the optimal choices for all features. The goal must then be to exploit *domain knowledge* to identify the most performant configuration. But since, as just explained, domain knowledge is unevenly developed and remains often incomplete, ExaStencils resorts to machine learning to derive a *feature-specific performance model* [80] that provides a comprehensible description of the effects of features and feature interactions on performance. The machine-learning techniques employ a set of configurations, the *learning set* or *sample set*, as input to learn from.

To this end, we have developed a *framework of configuration sampling and machine learning approaches* [36, 39, 59, 80] that allow us to derive a performance model of a given code that is *parameterized in terms of its features*. This way, we can express performance behavior in terms of concepts of the domain and determine automatically optimal configurations that are tailored to the problem at hand, which we have demonstrated in the domain of stencil codes [29, 30, 32] and beyond (e.g., databases, video encoders, compilers, and compression tools) [36, 39, 80]. Our framework integrates well with the other parts of ExaStencils that use and gather domain and configuration knowledge in different phases. It is similar in spirit to the performance modeling tool Extra-P, developed in the SPPEXA projects Catwalk und ExtraPeak [14], though we concentrate on flexibility in choosing and combining different sampling and learning techniques.

To demonstrate the usefulness of the machine-learning approach, we performed experiments on a large number of configurable software systems from different domains, among them different multigrid solvers including an implementation in ExaSlang [29, 30, 32]. To settle on a suitable learning set, we analyze the influence

of different sampling strategies on the accuracy of the performance-influence models [80].

Performance-influence models are not only meant to determine optimal configuration but also to *validate and refine domain knowledge*. We must be sure that the observed influences on the performance of the system match with our postulated knowledge, otherwise the system or the knowledge base must be revised. In a case study of a multigrid system working on triangular meshes, albeit not written in ExaSlang, we were able to validate the domain knowledge of the developers of the system [32].

For complex systems, learning a performance-influence model that captures accurately the performance behavior of all individual configurations is often time-consuming and the resulting models can become very large (i.e., many terms describing only a small influence on the performance). However, it is not always necessary to learn such a complex model; it might be beneficial to learn faster a simpler model that is less accurate but accurate enough for a given use-case [39]. In a set of experiments, we have demonstrated the tradeoff between accuracy, complexity, and time to learn a performance-influence model and shown that even simple models can predict all configurations with high accuracy. Depending on the use case of the performance-influence model, it can be of greater value to get quickly a rough impression of the most relevant influences of configuration options rather than waiting for a long time for a more detailed but also more complex model.

5 Case Studies

An essential ingredient of ExaStencils was a collection of case studies to test our approach for performance, target performance, and versatility. The more illustrative ones are sketched in this section. Section 5.1 deals with scalar elliptic PDEs, Sect. 5.2 with image processing applications and Sect. 5.3 with aspects of computational fluid dynamics. Section 5.4 details how generated solvers can be coupled with existing code, in our case molecular dynamics simulations. Finally, Sect. 5.5 goes beyond code generation and discusses a design decision study of porous media applications.

5.1 Scalar Elliptic Partial Differential Equations

A prominent and well-researched PDEs example that describes the steady state of the distribution of a physical quantity $u : \mathbb{R}^d \to \mathbb{R}$, such as heat in a solid medium, is

$$\begin{aligned} -\nabla \cdot (a \nabla u) &= f \quad \text{in} \quad \Omega\,, \\ u &= g \quad \text{on} \quad \partial\Omega \end{aligned}$$

www.dbooks.org

for a given domain Ω of dimensionality d, suitable boundary conditions, right-hand side $f : \mathbb{R}^d \to \mathbb{R}$ and a thermal conductivity $a : \mathbb{R}^d \to \mathbb{R}$ of the material to be simulated. Assuming a finite-difference discretization on a uniform grid, it can be observed that a constant a leads to a stencil with constant coefficients, whereas a variable a leads to one with variable coefficients. In the latter case, we always store all stencil coefficients, rather than values of a, unless specified otherwise. Lastly, if a is 1, the equation simplifies to the Poisson equation introduced in Sect. 3.2, which is given by $-\nabla^2 u = -\Delta u = f$.

In previous publications [43, 46, 48, 72], we demonstrated our ability to generate highly optimized and massively parallel geometric multigrid solvers for this model problem. For example, we achieve weak scalability up to the full *JUQUEEN* supercomputer, i.e., using up to 458,752 cores across 28,867 nodes, for moderately sized problems of 1–16 million unknowns per core. This is a good result considering that the ratio of data to be communicated, e.g., via MPI, is quite high compared to the required amount of computation. Figure 10 summarizes the obtained time to solution for up to $7.3 \cdot 10^{12}$ unknowns using a hybrid MPI/OpenMP parallelization for which one V(3, 3)-cycle takes about 3.6 s. A similar scaling behavior can also be observed in a scaling experiment on *Piz Daint* (see Fig. 11). With a largest problem of $7.3 \cdot 10^{12}$ unknowns solved, the ExaStencils software reaches for the Poisson equation a size comparable to the finite-element approach [27] developed in the TerraNeo project [8], where the solution of a stabilized tetrahedral linear finite-element discretization of a Stokes system is demonstrated with $1.7 \cdot 10^{13}$ unknowns. This is based on a Uzawa-type smoother [20] for the Stokes system. A first direct comparison has been made recently [41].

In a further single-node case study, we compared ExaStencils against the High-Performance Geometric Multigrid (HPGMG) benchmark [2]. Here, on one core of an Intel Xeon E5-2630 v2, the vectorized code generated by Athariac (see Sects. 4.2

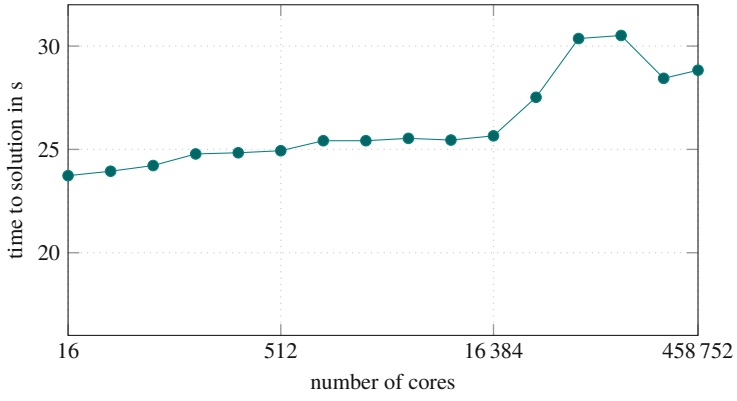


Fig. 10 Weak scaling for generated multigrid solvers using V(3, 3)-cycles to solve for Poisson's equation in 3D on *JUQUEEN* [47, 48]. The largest problem solved consists of $7.3 \cdot 10^{12}$ unknowns

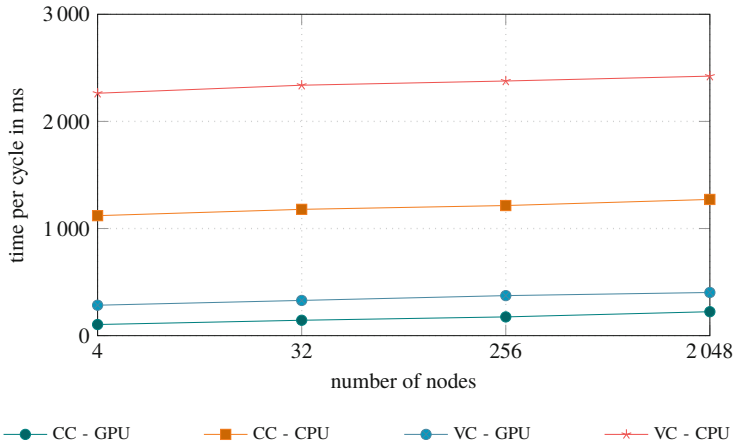


Fig. 11 Weak scaling on *Piz Daint* for constant coefficients (CC) and variable coefficients (VC). In all cases, the number of iterations required is nearly constant. Performance differences between CPU and GPU reflect the hardware characteristics

and 4.3) was able to solve 14.6 times as many unknowns per second as HPGMG. A closer inspection, using LIKWID [83], revealed that this performance gain is mainly due to a higher MFLOPS rate and a much better memory bandwidth exploitation of the solvers generated by ExaStencils. Further details, including also software productivity metrics (e.g., lines of code and Halstead complexity measures), are available elsewhere [76]. As an example, see also our case study on molecular dynamics (Sect. 5.4).

5.2 *Image Processing*

Next, we consider applications in variational image processing. State-of-the-art denoising algorithms based on total generalized variation have been implemented in ExaSlang [26] using a preconditioned Douglas-Rachford iteration for the underlying saddle-point problem. Results show a speedup of more than 4 compared to a reference Matlab implementation on CPU. As an indication of the capability of solving systems of PDEs as well, our most prominent imaging case study is the implementation of optical flow solvers.

Computation of the optical flow refers to the approximation of the apparent motion between two or more images which are part of an image sequence I. We used this application to illustrate ExaSlang's higher-dimensional data types [74].

Let us assume that $I(x, y, t)$ refers to a specific pixel (x, y) in an image sequence at time t, where t could refer to a certain frame in a video stream. Furthermore, we assume that a moving object's intensity does not change over time. Then, for small movements (corresponding to small time differences between two images), we may

www.dbooks.org

describe the movement of an intensity value at a pixel (x, y, t) as follows:

$$I(x, y, t) = I(x + dx, y + dy, t + dt)$$

After performing a Taylor expansion and reordering factors, we can define the partial image derivatives $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$ and $I_t = \frac{\partial I}{\partial t}$. This leads to the temporal gradient $\nabla_\theta I = (I_x, I_y, I_t)^T$ and the optical flow vector $(u, v) = \left(\frac{dx}{dt}, \frac{dy}{dt}\right)$, which can be transformed to the following system of PDEs to be solved:

$$\begin{aligned} -\alpha \Delta u + I_x(I_x u + I_y v) &= -I_x I_t \\ -\alpha \Delta v + I_y(I_x u + I_y v) &= -I_y I_t \end{aligned}$$

Here, α denotes the diffusion coefficient, which we set to 1 in this example. Furthermore, we set the time gradient I_t also to 1 for simplification purposes. After discretization, we obtain the following stencil:

$$\begin{pmatrix} & \begin{pmatrix} -\alpha & \\ & -\alpha \end{pmatrix} & \\ \begin{pmatrix} -\alpha & \\ & -\alpha \end{pmatrix} & \begin{pmatrix} 4\alpha + I_x^2 & I_x I_y \\ I_x I_y & 4\alpha + I_y^2 \end{pmatrix} & \begin{pmatrix} -\alpha & \\ & -\alpha \end{pmatrix} \\ & \begin{pmatrix} -\alpha & \\ & -\alpha \end{pmatrix} & \end{pmatrix}$$

This stencil can be directly mapped to Exaslang and used in a smoother kernel, as illustrated in Listing 14.

All corresponding variables, such as the unknowns u and v, are represented in ExaSlang as vectors of corresponding dimensionality. Thanks to our code generation approach, we can easily conduct parameter studies. For example, we varied the number of pre- and post-smoothing steps between 0 and 6 to study their influence on the convergence rate (see Christian Schmitt's dissertation [71] for details). Furthermore, we switched vectorization for Intel's AVX2 instruction set on and off. For our measurements, we used a single compute node equipped with an Intel i7-6700 CPU. We varied the number of threads between one (1T) and eight (8T) and used LIKWID [83] for our measurements, which is a supported back-end for ExaSlang's profiling capabilities.

A comparison of the energy consumption in relation to the application's execution time is depicted in Fig. 12. Clearly, the application is limited by the available memory bandwidth, as a speedup can be observed when going from one to two threads, but additional usage of CPU cores does not improve the speedup further. The use of AVX2 yields a small performance improvement compared to the scalar variants and, consequently, results in lower energy consumption. The V-cycle configurations that performed best in this case study are the V(4,2)-cycle, the V(3,3)-cycle, and the V(2,4)-cycle.

```
loop over fragments {
loop over Flow@current {
  Flow[next]@current = Flow[active]@current + (
    ( inverse ( diag ( SmootherStencil@current ) ) ) *
    ( RHS@current -
        SmootherStencil@current * Flow[active]@current )
    )
}}
advance Flow@current
```

Listing 14 Jacobi smoother definition using slots for the flow field

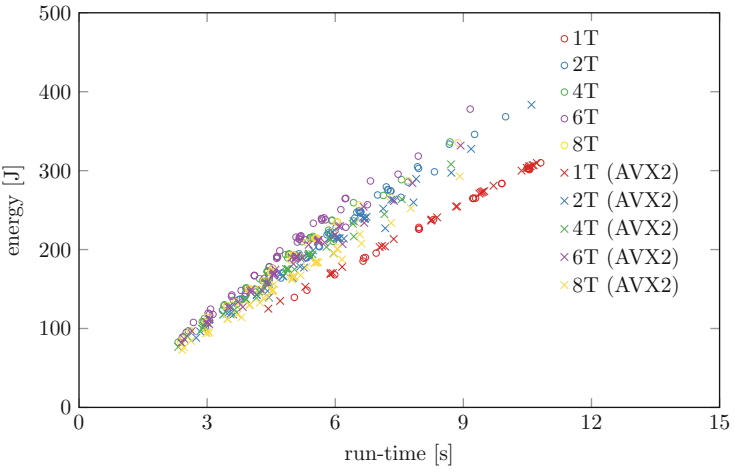


Fig. 12 Parameter study of different parallelization approaches and V-cycle configurations for the two-dimensional optical flow application

5.3 Computational Fluid Dynamics

Our next area of application is computational fluid dynamics (CFD), where we considered the solution of Stokes and Navier-Stokes equations [47, 50].

Let us turn first to the Stokes equations as introduced in Sect. 2.3. We discretized the given linear system of PDEs using finite differences and volumes on staggered grids. Additionally, more advanced multigrid components are required. This includes overlapping block smoothers that may additionally be colored. In ExaSlang, both can be expressed concisely and intuitively, as demonstrated in Sect. 4.1. Choosing a suitable coarse-grid solver is also a highly efficient process in our framework as it can be configured via our generate solver directive. Currently, we support conjugate gradient (CG), conjugate residual (CR), biconjugate gradient stabilized (BiCGSTAB) and minimal residual (MINRES), all with optional restart mechanisms, as well as simply applying the smoother. Of course, implementing

www.dbooks.org

one's own solver at layer 3 or layer 4 is still possible. We found that, in the collection of variants above, a BiCGSTAB with an added restart mechanism works reasonably well for the present test case [47].

Extending this work towards the full Navier-Stokes equations is the next step. For incompressible fluids, they can be given as

$$\frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u} \cdot \nabla)\,\boldsymbol{u} - \nu \nabla^2 \boldsymbol{u} + \frac{1}{\rho} \nabla p = \boldsymbol{f}_{\mathbf{u}}$$
$$\nabla \cdot \boldsymbol{u} = 0,$$

on a given domain Ω, which is square in 2D and cubic in 3D, with suitable boundary conditions and with $\boldsymbol{u}$, p and $\boldsymbol{f}_{\mathbf{u}}$ as before, ρ as the density and ν as the kinematic viscosity given by the ratio of the dynamic viscosity μ and ρ. The switch from Stokes to Navier-Stokes raises two challenges. First, since the equations become time-dependent, a suitable time integration is required. We choose a traditional implicit Euler, which can be implemented easily at layer 3 or layer 4. For each time step, one can then call either a generated solver variant or a custom implementation. Second, the employed solver has to be able to deal with non-linear equations. To this end, we allow expressing suitable linearizations, such as those based on Newton's method and Picard iterations [21, 47], at layer 2. The linearization chosen can be applied either locally or globally, and both variants can be expressed in tandem with the discretization approach at layer 2. Then, non-linear multigrid solvers based on the full approximation scheme can be tailored at layer 3 automatically. Scaling tests on the *JUWELS* supercomputer exhibit good strong scalability for $2.7 \cdot 10^8$ unknowns and good weak scalability on up to 24,576 cores, the largest amount available to us at the time of the experiment [47]. The results are summarized in Figs. 13 and 14.

This approach can also be extended to the scope of our most sophisticated application, namely the simulation of non-Newtonian and non-isothermal fluids. Originally, we ported a legacy FORTRAN code to ExaSlang 4 [50] resulting not only in a considerable reduction in code complexity and size of about one order of magnitude, but also in a reasonable gain in performance of also about one order of magnitude on average [50]. One prerequisite is the support of staggered non-uniform grids by our code generation framework and DSL, which we extended accordingly. After implementing fully parallel grid setup routines, we are now also able to generate applications executable on GPUs and cluster architectures from the same ExaSlang code. This application also serves as a motivation to extend our generator to apply domain-specific optimizations, such as our sophisticated loop-carried common subexpression elimination [45], which are able to improve performance even further. Finally, using the previously described layer 2 and layer 3 capabilities allows for even more expressive and concise specifications, which have moreover a greater ability to reflect concepts familiar to domain scientists.

Lastly, we should mention that we invested some time to explore alternative approaches, such as the Lattice-Boltzmann method, which can also be implemented

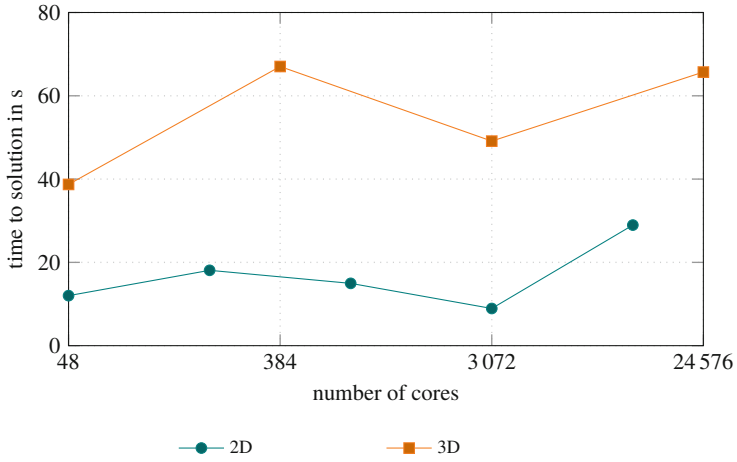


Fig. 13 Weak scaling of generated multigrid solvers for the Navier-Stokes equations in 2D and 3D on *JUWELS* [47]

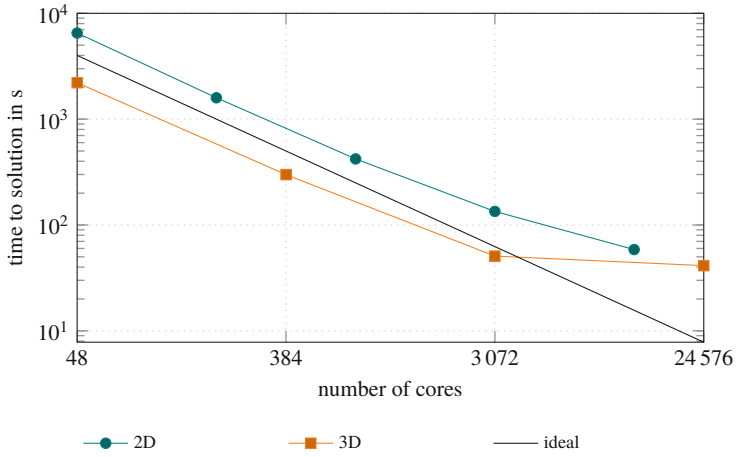


Fig. 14 Strong scaling of generated multigrid solvers for the Navier-Stokes equations in 2D and 3D on *JUWELS* [47]

in our DSL [64]. An early evaluation showed that applications generated with the ExaStencils framework are about a factor of 2 slower compared to the state-of-the-art multiphysics framework waLBerla.[5]

[5] www.walberla.net.

www.dbooks.org

5.4 Molecular Dynamics Simulation

To illustrate the versatility of the ExaStencils approach, we demonstrated that the ExaStencils compiler could be used to generate a multigrid solver that integrates into a large, existing simulation code. For this purpose, we considered a molecular dynamics simulation code used in practice [67, 68], a simulation that computes the motion of the electrons and nuclei of molecules or groups of molecules.

The simulation is a so-called *real-space density functional theory* (RSDFT) simulation. A simulation in real space computes the wave functions on a discretized Cartesian grid, which avoids costly computations of the fast Fourier transform (FFT) that methods which work in Fourier space have to perform. The simulation computes the motion of the particles using the Kohn-Sham density functional theory, which is implemented using a Car-Parrinello molecular dynamics scheme.

The RSDFT simulation requires the computation of the electron-electron Coulomb interaction potential of the charge density. This potential is computed efficiently by solving the corresponding Poisson equation using a multigrid method. The RSDFT code contains a manually derived multigrid solver, which we wanted to replace by a solver generated by the ExaStencils compiler.

However, integrating a solver into an existing code base requires some additional steps above the ones necessary to generate a stand-alone solver. The RSDFT code has its own set of grid data structures for storage of the wave functions. To obtain optimal performance, the ExaStencils compiler generates a set of internal data structures that are the most suitable for the computation (see Sect. 4.3). These data structures are usually not compatible with the ones an application code uses. While it would be possible to force the compiler to use the RSDFT data structures, it would deprive the compiler of some of its optimization opportunities. As a workaround, we used the compiler to generate a set of data transfer routines that copy the grid functions from the RSDFT data structures to the internal ones and vice versa.

In Table 1, we compare the performance of the automatically generated multigrid solver and the legacy solver that was manually written in C. Note that both solvers implement different coarse-grid solution strategies. Thus, the overall timings are not directly comparable, because the number of iterations change between the implementations. However, the time per iteration is indicative for the quality of the code of the generated solver, and we see that it is comparable to the time per iteration of the legacy code. The generated code is often slightly faster.

We can conclude that the ExaStencils approach is also feasible in situations that involve an already existing codebase.

5.5 Porous Media

A further study, in cooperation with SPPEXA project EXA-DUNE [7], concerned the design decision made for stencil codes in the domain of porous media [31]. The goal of this study was to help application engineers understand the complexity

Table 1 Timings of the solution of the Poisson equation discretized on a grid of n points on *JURECA* [42] using the legacy solver and an automatically generated multigrid solver

	Legacy			ExaStencils		
Cores	Iterations	Time	Time/iteration	Iterations	Time	Time/iteration
$n = 127^3$						
1	6	0.87 s	1.4×10^{-1} s	6	0.76 s	1.3×10^{-1} s
2	6	0.51 s	8.5×10^{-2} s	6	0.39 s	6.4×10^{-2} s
4	6	0.31 s	5.1×10^{-2} s	6	0.27 s	4.5×10^{-2} s
8	6	0.27 s	4.4×10^{-2} s	6	0.13 s	2.1×10^{-2} s
$n = 255^3$						
1	28	32.3 s	1.15 s	6	6.00 s	1.00 s
2	28	17.2 s	0.61 s	6	3.01 s	0.50 s
4	28	9.70 s	0.35 s	6	1.67 s	0.28 s
8	28	5.72 s	0.20 s	6	1.01 s	0.17 s
16	28	3.86 s	0.14 s	6	0.84 s	0.14 s
24	28	2.94 s	0.10 s	6	0.44 s	7.3×10^{-2} s

of stencil computations and the relation between the mathematical model of an application and the stencil used to solve the partial differential equation underlying the application. To model these design decisions, we use feature orientation as introduced in Sect. 4.6. An example of a design decision is the choice of the type of boundary conditions that have to be satisfied by a given application. We differentiate between decisions at the mathematical and the stencil level, and we devise one feature model for each of these levels. Since decisions at the mathematical level may affect decisions that can be made at the stencil level, we also include constraints between these two levels.

To demonstrate the usefulness of feature modeling, we considered a set of applications that deviate from the standard 5-point stencil used in many benchmarks. We demonstrated that feature models can be used beyond simple mathematical problems by considering problems of different complexity. Specifically, we started with the simple heat equation and went on to more complex equations, such as the advection-diffusion equation with operator splitting or Richard's equation. For all of these equations, we were able to express the design decisions at the mathematical and the stencil level as well as their dependences. The result is a comprehensive set of feature models that can be uses to formally reason about the variabilities in stencil codes (e.g., to identify dependencies among configuration decisions).

www.dbooks.org

6 Variants of the ExaStencils Approach

6.1 ExaSlang 4 Embedded, Not External

In the second funding period, ExaStencils was part of an international collaboration with Shigeru Chiba, co-funded by the Japan Science and Technology Agency (JST). This project has already been reported to and reviewed by JST as part of the Japanese strategic basic research programme CREST [17]. The summary is also available at the author's Web site.[6]

The project aimed at a software architecture for embedded domain-specific languages. The cooperation of the DFG and JST in SPPEXA gave Chiba's team the opportunity to explore the applicability of its software architecture in the context of ExaStencils. ExaSlang 4 is a stand-alone language that needs its own compiler and development environments, such as an editor and a debugger, and that has a relatively large development cost. The principle of embedding is an approach to reduce the size and cost of the infrastructure needed. An embedded DSL is implemented on top of a general-purpose host language as a library with a programming interface that looks like a language. Since an embedded DSL is a library, its development cost is usually smaller and programmers can reuse the environments of the host language.

The result of the project was an embedded-DSL architecture based on run-time metaprogramming. In this architecture, a part of the running program is *reified* at run time; that is, its abstract syntax tree is dynamically extracted. Then it is translated to efficient binary code to be run. For example, the host language could be Java, while the program is translated to C/C++ and compiled to machine code. The idea is to view the extracted code as a program not in the host language, but in the DSL with a slightly different semantics. Hence, we can apply domain-specific optimization, such as partial evaluation, during translation.

To evaluate this architecture, Chiba's team developed two software systems. One is Bytespresso [16], which they used to implement an embedded DSL that mimics Exaslang 4. They were successful in developing the DSL with a similar programming interface and then measured the execution performance. Although the computational part alone experienced a loss by a factor of 3, the total execution including compilation was twice as fast in the largest case measured: 12 grid levels,[7] which is still small-scale. This demonstrates that, for small-scale problems, the embedded version of Exaslang 4 is more suitable for interactive execution than the external version.

[6] https://post-peta-crest.github.io/chiba/.

[7] Figures of the measurements can be found in the CREST report [17].

```
Fragments.loop_over do
  Flow_u.current.loop_over do
    Flow_u[:next].current = Flow_u[:active].current +
     ((1.0 / diag(SmootherStencil_u.current)) *
      (RHS_u.current - SmootherStencil_u.current *
            Flow_u[:active].current))
  end
end
```

Listing 15 Specification of an optical flow solver in Ruby-embedded ExaSlang 4

Chiba's team also explored the expressive power of the architecture it proposed in Yadriggy [15], a framework for embedding DSLs in Ruby. Compared to the ExaSlang 4 implementation in Java using Bytespresso, exhibiting a source code syntax far from the original, the Ruby-embedded version on Yadriggy features a syntax much closer to the original. This is due to the flexible syntax of Ruby and the development support by Yadriggy. For example, the code fragment depicted in Listing 15 is ExaSlang 4 code taken from the optical flow case study (Sect. 5.2, Listing 14) embedded in Ruby. This internal DSL code is not interpreted as Ruby code and, hence, the language does not have to adhere to Ruby's semantics.

6.2 A Multigrid Solver in SPIRAL

Forerunner and motivator for the vision of ExaStencils was the US project SPIRAL [24, 61]. SPIRAL's initial and foremost domain has been linear transformations. Recently it went on to small-scale linear algebra [81]. ExaStencils' domain is a subdomain of multigrid computations. At project halftime, we put a very simple case of multigrid on SPIRAL: a solver with a Richardson smoother for a discretized square 2D Poisson equation with Dirichlet boundary conditions [11]. The central step is to bring the smoother into an algebraic form of about a dozen rewrite equations. It was an effort of a few days, but the present implementation of SPIRAL does not support the broader domain of ExaStencils. For the generation of efficient code, special adaptations are required as illustrated in the Bachelor thesis of Sebastian Schweikl [79]. This is a consequence of the fact that SPIRAL's target domain has been signal processing and not the solution of PDEs. The applicability of SPIRAL in its present form to our domain is very limited.

www.dbooks.org

7 The Legacy of ExaStencils

7.1 Effort

ExaStencils was not the only project in SPPEXA to address the solution of PDEs via multigrid methods. However, it was the only one that took a revolutionary rather than evolutionary software approach, i.e., that did not build on existing software tools or application software but started afresh with the automatic, domain-specific synthesis and optimization of application software via a set of dedicated domain-aware software tools. The challenge was to identify the domain knowledge that is useful in helping an optimization, to build the domain-aware tools and to generate some target codes that demonstrate that this approach is realistic and promising.

A central ingredient of the approach is the stratification of the DSL ExaSlang into four layers of abstraction, each addressing a different set of aspects of the stencil code. The domain knowledge is twofold. Knowledge of the application problem is provided by hints that enable the ExaSlang compiler to optimize at the next-more concrete layer. These hints can be supplied as code annotations or conceptually by other means, e.g., configuration files or GUIs, as is the case in the ExaSlang Level 1 editor documented in Tim Ammenhäuser's Bachelor's thesis [4]. Knowledge of the execution platform is provided by a dedicated platform description language.

The main effort in the project concerned the design and implementation of the ExaSlang compiler and code generator which had to be built from scratch, since ExaSlang is an external DSL. This effort led to three dissertations of the six that emerged from the project [43, 47, 71] and two habilitation theses [35, 40]. In a sideline, a comparative study was made with an internal DSL mimicking the most concrete layer of the external ExaSlang, which proved the approach doable but not the first choice for exascale software.

Another major effort was to drive the development of performance prediction further. On the software side, this will soon be realized by one cumulative dissertation introducing the, also revolutionary, generation of performance prediction models via machine learning in the setting of software product line engineering [28]. On the mathematics side, a dissertation drove local Fourier analysis for the convergence prediction of multigrid codes further [66]. Another dissertation in the realm of multigrid math drove the technology of multigrid smoothers forward [19].

The final major element was the case studies conducted to illustrate the impact of the ExaStencils approach.

7.2 Outreach

Let us sketch how ExaStencils profited from and nurtured other research projects.

There was a bilateral exchange of ideas and best practices with SPPEXA project TerraNeo [8] that deals with multigrid solvers on block-structured grids

for applications in geoscience. Both ExaStencils and TerraNeo have dealt in loose collaboration with high-performance multigrid codes. One result was an evaluation from the perspective of code generation technology [41]. A further systematic comparison of the performance regarding the algorithms, data structures and their implementation is left to the future. However, the generative programming technologies of ExaStencils are already being leveraged for the redesign of the TerraNeo code basis [38], where they help to simplify the software structure and to reduce the coding effort.

In cooperation with SPPEXA project EXA-DUNE [7], we developed a semi-automatic variability extraction approach that generates a family of applications based on a given application [31]. An ExaStencils-generated application solves the same problem as the EXA-DUNE application but uses sets of methods different from the DUNE framework [9] with the goal of optimizing performance. For example, applications might use different grid implementations or different finite-element maps provided by the DUNE framework. The goal of this approach is to ease the burden of having to understand the DUNE framework when adapting a given application to new use cases and optimizing performance. In a first evaluation, the approach was able to identify over 90% of the alternative but compatible methods a developer of the framework has identified. Using this automated approach, we were even able to identify a bug and an inconsistency in the DUNE framework.

A molecular dynamics simulation on the application platform RSDFT [67] profited from the ExaStencils approach. The port of a development in ExaSlang to RSDFT was successful.

Applied researchers in triangular meshes profited from the ExaStencils approach of validating and refining domain knowledge via a performance-influence model [32]. In simpler words, an automatic learning procedure confirmed their design choices and their effects on performance and made them aware of others.

In a technically related BMBF-funded project, called HighPerMeshes, which also aims at creating a DSL but for the scientific-computing domain of unstructured meshes [37], the parallelization technology SYCL was evaluated [3]. This fueled our interest in creating a SYCL back-end for our code generator which, in turn, provided valuable insights into possible code generation strategies and optimizations for project HighPerMeshes.

SPIRAL [24, 61] was ExaStencils' main motivator and an orientation point for positioning its contribution [11]. At present, SPIRAL is not as serious a platform for multigrid solver development as ExaStencils and it would support the algebraic rather than the geometric form of multigrid.

Similarly, HIPA^{cc} [55, 57], a DSL embedded in C++ and a source-to-source translator for image processing applications that can target a wide variety of parallel accelerator architectures, inspired us on how to deal with both domain and architecture knowledge. In turn, at the start of ExaStencils, we introduced language constructs to HIPA^{cc} to process and represent data at different resolutions, which enables the specification of applications that work on image pyramids as well as 2D stencil-based multigrid methods. By decoupling the algorithm from its schedule,

www.dbooks.org

HIPA^{cc} allows the generation of efficient stencil-code implementations for single accelerators [56].

A recent DFG-funded project on ocean modeling (grant. no 320126236) challenges the extensibility of ExaStencils' application domain by considering time-dependent hyperbolic partial differential equations, discretized using finite-volume or higher-order discontinuous Galerkin methods on block-structured triangular grids. Since, so far, an explicit time-stepping scheme is being applied, no solvers like multigrid are required. However, it has been shown already that the ExaStencils framework is capable of generating efficient and scalable code for such applications on GPU clusters [49].

7.3 Potential

The ExaSlang compiler and code generator remain at the stage of a prototype, but we believe that they have demonstrated that a stratified optimizing DSL approach can provide dramatically increased convenience for the application programmer and high flexibility concerning the execution platform. The effort of adapting a complex code to a different application or platform can be reduced significantly. The price to be paid is good knowledge of the theory on which the application rests and a significant development cost in implementing the DSL. The latter can be reduced by going the internal rather than the external way, at the price of reduced expressiveness and flexibility in terms of target code optimization.

Our choice not to go with an embedded DSL tends to be not popular in present-day DSL development but is essential. We did not want to be bound by any language limitations in exploring the potential of the ExaStencils approach to domain-specific programming and optimization. To achieve high portability and execution performance of generated programs, we selected C/C++ as the target language.

Our case studies covered the range from textbook examples to realistic applications. The latter are represented by a simulation of non-Newtonian flow, fully developed in the ExaSlang world, and a molecular dynamics simulation which used an ExaStencils solver through an automatically generated compatibility interface. We have demonstrated that we can reach the expected performance, achieve portability to most contemporary HPC platforms, and profit from a substantial decrease of development time of new PDE models that can be expressed in ExaSlang.

Since the start of our project, code generation has become a standard technique in many HPC codes and, thus, the concepts developed in ExaStencils are used or can be adapted to other domains. Currently, a popular approach is the generation of single compute kernels from an embedded DSL for existing software frameworks. In addition, C++ template-based embedded DSLs are common. Frequently, both the concrete DSL and the resulting implementations are very application-specific, but the intermediate representations and code transforms can be defined quite generally and then optimized for the specific case. In the future, it should be possible to

extend the ExaStencils approach to other HPC applications to enable more holistic optimizations than currently established approaches.

Acknowledgments Project ExaStencils received generous funding from the Deutsche Forschungsgemeinschaft (DFG) in its priority programme SPP 1648 "Software for Exascale Computing" under grant numbers AP 206/7, BO 3405/2, LE 912/15, RU 422/15 and TE 163/17. We thank the Jülich Supercomputing Center (JSC) for providing access to the supercomputers *JUQUEEN* and *JURECA* and the Swiss National Supercomputing Centre (CSCS) for providing computational resources and access to the supercomputer *Piz Daint*. We are grateful to Rochus Schmid for letting us have the RSDFT code for the molecular dynamics simulation.

References

1. Abelson, H., Sussman, G.J., Sussman, J.: Structure and Interpretation of Computer Programs, 2nd edn. The MIT Press, Cambridge (1996)
2. Adams, M.F., Brown, J., Shalf, J., Straalen, B.V., Strohmaier, E., Williams, S.: HPGMG 1.0: a benchmark for ranking high performance computing systems. Tech. Rep. LBNL-6630E, Lawrence Berkeley National Laboratory (2014)
3. Afzal, A., Schmitt, C., Alhaddad, S., Grynko, Y., Teich, J., Förstner, J., Hannig, F.: Solving Maxwell's equations with modern C++ and SYCL: a case study. In: Proceedings of the Annual IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP), pp. 49–56. IEEE, New York (2018)
4. Ammenhäuser, T.: Online-editor und visualisierung für ExaStencils. Bachelor's thesis, University of Wuppertal (2019)
5. Apel, S., Batory, D., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines: Concepts and Implementation. Springer, Berlin (2013)
6. Bandishti, V., Pananilath, I., Bondhugula, U.: Tiling stencil computations to maximize parallelism. In: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC), pp. 40:1–40:11. IEEE Computer Society, Washington (2012)
7. Bastian, P., Altenbernd, M., Dreier, N.A., Engwer, C., Fahlke, J., Fritze, R., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Turek, S.: EXA-DUNE: flexible PDE solvers, numerical methods and applications (in this volume)
8. Bauer, S., Bunge, H.P., Drzisga, D., Ghelichkhan, S., Huber, M., Kohl, N., Mohr, M., Rüde, U., Thönnes, D., Wohlmuth, B.: TerraNeo: mantle convection beyond a trillion degrees of freedom (in this volume)
9. Blatt, M., Burchardt, A., Dedner, A., Engwer, C., Fahlke, J., Flemisch, B., Gersbacher, C., Gräser, C., Gruber, F., Grüninger, C., Kempf, D., Klöfkorn, R., Malkmus, T., Müthing, S., Nolte, M., Piatkowski, M., Sander, O.: The distributed and unified numerics environment, version 2.4. Arch. Numer. Softw. **4**(100), 13–29 (2016)
10. Bolten, M., Rittich, H.: Fourier analysis of periodic stencils in multigrid methods. SIAM J. Sci. Comput. **40**(3), A1642–A1668 (2018)
11. Bolten, M., Franchetti, F., Kelly, P.H.J., Lengauer, C., Mohr, M.: Algebraic description and automatic generation of multigrid methods in SPIRAL. Concurr. Comput. Pract. Exp. **29**(17), 4105:1–4105:11 (2017). Special issue: Advanced Stencil-Code Engineering
12. Bondhugula, U., Hartono, A., Ramanujam, J., Sadayappan, P.: A practical automatic polyhedral parallelizer and locality optimizer. In: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), pp. 101–113. ACM, New York (2008)

www.dbooks.org

13. Bondhugula, U., Bandishti, V., Pananilath, I.: Diamond tiling: tiling techniques to maximize parallelism for stencil computations. IEEE Trans. Parallel Distrib. Syst. **28**(5), 1285–1298 (2017)
14. Calotoiu, A., Copik, M., Hoefler, T., Ritter, M., Wolf, F.: ExtraPeak: advanced automatic performance modeling for HPC applications (in this volume)
15. Chiba, S.: Foreign language interfaces by code migration. In: 18th ACM SIGPLAN International Conference on Generative Programming: Concepts & Experiences (GPCE), pp. 1–13. ACM, New York (2019)
16. Chiba, S., Zhuang, Y., Scherr, M.: Deeply reifying running code for constructing a domain-specific language. In: Proceedings of the 13th International Conference on Principles and Practices of Programming on the Java Platform: Virtual Machines, Languages, and Tools (PPPJ), pp. 1:1–1:12. ACM, New York (2016)
17. Chiba, S., Zhuang, Y., Dao, T.C.: A development platform for embedded domain-specific languages. In: Sato, M. (ed.) Advanced Software Technologies for Post-Peta Scale Computing, chap. 8, pp. 139–161. Springer, Singapore (2019)
18. Christen, M., Schenk, O., Burkhart, H.: PATUS: a code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures. In: Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 676–687 (2011)
19. Claus, L.: Multigrid smoothers for saddle point systems. Ph.D. thesis, University of Wuppertal (2019). 133 pp.
20. Drzisga, D., John, L., Rüde, U., Wohlmuth, B., Zulehner, W.: On the analysis of block smoothers for saddle point problems. SIAM J. Matrix Anal. Appl. **39**(2), 932–960 (2018)
21. Elman, H.C., Silvester, D.J., Wathen, A.J.: Finite Elements and Fast Iterative Solvers. Numerical Mathematics and Scientific Computation, 2nd edn. Oxford University Press, Oxford (2014)
22. Feautrier, P., Lengauer, C.: Polyhedron model. In: Padua, D.A., et al. (eds.) Encyclopedia of Parallel Computing, pp. 1581–1592. Springer, New York (2011)
23. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: evolutionary algorithms made easy. J. Mach. Learn. Res. **13**, 2171–2175 (2012)
24. Franchetti, F., Low, T.M., Popovici, D.T., Veras, R.M., Spampinato, D.G., Johnson, J.R., Püschel, M., Hoe, J.C., Moura, J.M.F.: SPIRAL: extreme performance portability. Proc. IEEE **106**(11), 1935–1968 (2018). Special issue: From High-Level Specification to High-Performance Code
25. Frigo, M., Johnson, S.G.: The design and implementation of FFTW3. Proc. IEEE **93**(2), 216–231 (2005). Special issue: Program Generation, Optimization, and Platform Adaptation
26. Gerecke, M.: Implementierung des TGV Algorithmus mithilfe von ExaSlang. Bachelor's thesis, Friedrich-Alexander University Erlangen-Nürnberg (2017)
27. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. J. Computat. Sci. **17**, 509–521 (2016)
28. Grebhahn, A.: Performance prediction of highly configurable software systems: multigrid systems and beyond. Ph.D. thesis, Faculty of Computer Science and Mathematics, University of Passau (2020, to be submitted)
29. Grebhahn, A., Kuckuk, S., Schmitt, C., Köstler, H., Siegmund, N., Apel, S., Hannig, F., Teich, J.: Experiments on optimizing the performance of stencil codes with SPL Conqueror. Par. Proc. Lett. **24**(3), 19 pp. (2014). Article 1441001
30. Grebhahn, A., Siegmund, N., Köstler, H., Apel, S.: Performance prediction of multigrid-solver configurations. In: Software for Exascale Computing – SPPEXA 2013–2015. Lecture Notes in Computational Science and Engineering, vol. 113, pp. 69–88. Springer, New York (2016)
31. Grebhahn, A., Engwer, C., Bolten, M., Apel, S.: Variability of stencil computations for porous media. Concurr. Computat. Pract. Exp. **29**(17), 4119:1–4119:14 (2017). Special issue: Advanced Stencil-Code Engineering

32. Grebhahn, A., Rodrigo, C., Siegmund, N., Gaspar, F.J., Apel, S.: Performance-influence models of multigrid methods: a case study on triangular meshes. Concurr. Comput. Pract. Exp. **29**(17), 4057:1–4057:13 (2017). Special issue: Advanced Stencil-Code Engineering
33. Groth, S., Schmitt, C., Teich, J., Hannig, F.: SYCL code generation for multigrid methods. In: Proceedings of the 22nd International Workshop on Software and Compilers for Embedded Systems (SCOPES), pp. 41–44. ACM, New York (2019)
34. Gysi, T., Osuna, C., Fuhrer, O., Bianco, M., Schulthess, T.C.: Stella: a domain-specific tool for structured grid methods in weather and climate models. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), pp. 41:1–41:12. ACM, New York (2015)
35. Hannig, F.: Domain-specific and resource-aware computing (2017). Habilitation thesis, Friedrich-Alexander University Erlangen-Nürnberg, 444 pp.
36. Kaltenecker, C., Grebhahn, A., Siegmund, N., Guo, J., Apel, S.: Distance-based sampling of software configuration spaces. In: Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE), pp. 1084–1094. IEEE Computer Society, Washington (2019)
37. Kenter, T., Mahale, G., Alhaddad, S., Grynko, Y., Schmitt, C., Afzal, A., Hannig, F., Förstner, J., Plessl, C.: OpenCL-based FPGA design to accelerate the nodal discontinuous Galerkin method for unstructured meshes. In: Proceedings of the 26th IEEE International Symposium on Field-Programmable Custom Computing Machines (FCCM), pp. 189–196. IEEE, New York (2018)
38. Kohl, N., Thönnes, D., Drzisga, D., Bartuschat, D., Rüde, U.: The HyTeG finite-element software framework for scalable multigrid solvers. Int. J. Parallel Emergent Distrib. Syst. 1–20 (2018).
39. Kolesnikov, S., Siegmund, N., Kästner, C., Grebhahn, A., Apel, S.: Tradeoffs in modeling performance of highly-configurable software systems. Softw. Syst. Model. **18**(3), 2265–2283 (2019)
40. Köstler, H.: Effiziente numerische Algorithmen und Softwareentwicklung für hochparallele Rechensysteme (2014). Habilitation thesis, Friedrich-Alexander University Erlangen-Nürnberg, 94 pp.
41. Köstler, H., Heisig, M., Kohl, N., Kuckuk, S., Bauer, M., Rüde, U.: Code generation approaches for parallel geometric multigrid solvers. Analele Stiintifice ale Universitatii Ovidius, Seria Matematica (2019, accepted for publication)
42. Krause, D., Thörnig, P.: JURECA: modular supercomputer at Jülich Supercomputing Centre. J. Large-Scale Res. Fac. **4**, A132, 9 pp. (2018)
43. Kronawitter, S.: Automatic optimization of stencil codes. Ph.D. thesis, University of Passau (2019), 130 pp.
44. Kronawitter, S., Lengauer, C.: Polyhedral search space exploration in the ExaStencils code generator. ACM Trans. Archit. Code Op. **15**(4), 40:1–40:25 (2019)
45. Kronawitter, S., Kuckuk, S., Lengauer, C.: Redundancy elimination in the ExaStencils code generator. In: Carretero, J., et al. (eds.) Algorithms and Architectures for Parallel Processing (ICA3PP), Collocated Workshops. Lecture Notes in Computer Science, vol. 10049, pp. 159–173. Springer, New York (2016). First International Workshop on Data Locality in Modern Computing Systems (DLMCS)
46. Kronawitter, S., Kuckuk, S., Köstler, H., Lengauer, C.: Automatic data layout transformations in the ExaStencils code generator. Parallel Proc. Lett. **28**(3), 18 pp. (2018). Article 1850009
47. Kuckuk, S.: Automatic code generation for massively parallel applications in computational fluid dynamics. Ph.D. thesis, Friedrich–Alexander-Universität Erlangen-Nürnberg (2019), 243 pp.
48. Kuckuk, S., Köstler, H.: Automatic generation of massively parallel codes from ExaSlang. Computation **4**(3), 20 pp. (2016). Article 27. Special issue: High Performance Computing (HPC) Software Design
49. Kuckuk, S., Köstler, H.: Whole program generation of massively parallel shallow water equation solvers. In: 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 78–87. IEEE, New York (2018)

www.dbooks.org

50. Kuckuk, S., Haase, G., Vasco, D., Köstler, H.: Towards generating efficient flow solvers with the ExaStencils approach. Concurr. Comput. Pract. Exp. **29**(17), 4062:1–4062:17 (2017). Special issue: Advanced Stencil-Code Engineering
51. Kuckuk, S., Leitenmaier, L., Schmitt, C., Schönwetter, D., Köstler, H., Fey, D.: Towards virtual hardware prototyping for generated geometric multigrid solvers. Tech. Rep. CS 2017-01, Department of Computer Science, Friedrich-Alexander University Erlangen-Nürnberg (2017). 10 pp.
52. Lange, M., Kukreja, N., Louboutin, M., Luporini, F., Vieira, F., Pandolfo, V., Velesko, P., Kazakas, P., Gorman, G.: Devito: towards a generic finite difference DSL using symbolic Python. In: Proceedings of the Workshop on Python for High-Performance and Scientific Computing (PyHPC), pp. 67–75. IEEE, New York (2016)
53. Lengauer, C., Apel, S., Bolten, M., Größlinger, A., Hannig, F., Köstler, H., Rüde, U., Teich, J., Grebhahn, A., Kronawitter, S., Kuckuk, S., Rittich, H., Schmitt, C.: ExaStencils: advanced stencil-code engineering. In: Lopes, L., et al. (eds.) Euro-Par 2014: Parallel Processing Workshops, Part II. Lecture Notes in Computer Science, vol. 8806, pp. 553–564. Springer, New York (2014)
54. Luporini, F., Varbanescu, A.L., Rathgeber, F., Bercea, G.T., Ramanujam, J., Ham, D.A., Kelly, P.H.J.: Cross-loop optimization of arithmetic intensity for finite element local assembly. ACM Trans. Archit. Code Op. **11**(4), 57:1–57:25 (2015)
55. Membarth, R., Hannig, F., Teich, J., Körner, M., Eckert, W.: Generating device-specific GPU code for local operators in medical imaging. In: Proceedings of the 26th IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 569–581. IEEE, New York (2012)
56. Membarth, R., Reiche, O., Schmitt, C., Hannig, F., Teich, J., Stürmer, M., Köstler, H.: Towards a performance-portable description of geometric multigrid algorithms using a domain-specific language. J. Parallel Distrib. Comput. **74**(12), 3191–3201 (2014)
57. Membarth, R., Reiche, O., Hannig, F., Teich, J., Körner, M., Eckert, W.: HIPA^{cc}: a domain-specific language and compiler for image processing. IEEE Trans. Parallel Distrib. Syst. **27**(1), 210–224 (2016)
58. Mullapudi, R.T., Vasista, V., Bondhugula, U.: PolyMage: automatic optimization for image processing pipelines. SIGARCH Comput. Archit. News **43**(1), 429–443 (2015)
59. Nair, V., Yu, Z., Menzies, T., Siegmund, N., Apel, S.: Finding faster configurations using FLASH. IEEE Trans. Softw. Eng. (2018) [Online first]
60. Padua, D.A., et al. (eds.): Encyclopedia of Parallel Computing. Springer, New York (2011)
61. Püschel, M., Franchetti, F., Voronenko, Y.: Spiral. In: Padua, D.A., et al. (eds.) Encyclopedia of Parallel Computing, pp. 1920–1933. Springer, New York (2011)
62. Ragan-Kelley Jonathan, E.A.: Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. ACM SIGPLAN Not. **48**(6), 519–530 (2013). Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)
63. Rathgeber, F., Ham, D.A., Mitchell, L., Lange, M., Luporini, F., Mcrae, A.T.T., Bercea, G.T., Markall, G.R., Kelly, P.H.J.: Firedrake: automating the finite element method by composing abstractions. ACM Trans. Math. Softw. **43**(3), 24:1–24:27 (2017)
64. Ribica, D.: Code generation vs. HPC framework. Bachelor's thesis, Friedrich-Alexander University Erlangen-Nürnberg (2018)
65. Rittich, H.: LFA Lab. https://hrittich.github.io/lfa-lab/
66. Rittich, H.: Extending and automating Fourier analysis for multigrid methods. Ph.D. thesis, Faculty of Mathematics and Natural Sciences, University of Wuppertal (2017). 202 pp.
67. Schmid, R.: Car-Parrinello simulations with a real space method. J. Comput. Chem. **25**(6), 799–812 (2004)
68. Schmid, R., Tafipolsky, M., König, P.H., Köstler, H.: Car-Parrinello molecular dynamics using real space wavefunctions. Phys. Status Solidi (B) **243**(5), 1001–1015 (2006)

69. Schmid, M., Reiche, O., Schmitt, C., Hannig, F., Teich, J.: Code generation for high-level synthesis of multiresolution applications on FPGAs. In: Proceedings of the First International Workshop on FPGAs for Software Programmers (FSP), pp. 21–26 (2014). arXiv:1408.4721
70. Schmid, M., Schmitt, C., Hannig, F., Malazgirt, G.A., Sönmez, N., Yurdakul, A., Cristal, A.: Big Data and HPC acceleration with Vivado HLS. In: Koch, D., Hannig, F., Ziener, D. (eds.) FPGAs for Software Programmers, chap. 7, pp. 115–136. Springer, New York (2016)
71. Schmitt, C.: A domain-specific language and source-to-source compilation framework for geometric multigrid methods. Ph.D. thesis, Friedrich-Alexander University Erlangen-Nürnberg (2019). Verlag Dr. Hut, 203 pp.
72. Schmitt, C., Kuckuk, S., Hannig, F., Köstler, H., Teich, J.: ExaSlang: a domain-specific language for highly scalable multigrid solvers. In: Proceedings of the 4th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), pp. 42–51. IEEE Computer Society, Washington (2014)
73. Schmitt, C., Schmid, M., Hannig, F., Teich, J., Kuckuk, S., Köstler, H.: Generation of multigrid-based numerical solvers for FPGA accelerators. In: Größlinger, A., Köstler, H. (eds.) Proceedings of the 2nd International Workshop on High-Performance Stencil Computations (HiStencils), pp. 9–15 (2015). www.viaprinto.de
74. Schmitt, C., Kuckuk, S., Hannig, F., Teich, J., Köstler, H., Rüde, U., Lengauer, C.: Systems of partial differential equations in ExaSlang. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing – SPPEXA 2013-2015, Lecture Notes in Computational Science and Engineering, vol. 113, pp. 47–67. Springer, Heidelberg (2016)
75. Schmitt, C., Hannig, F., Teich, J.: A target platform description language for code generation in HPC. In: Workshop Proceedings of the 31st GI/ITG International Conference on Architecture of Computing Systems (ARCS), pp. 59–66. VDE, Berlin (2018)
76. Schmitt, C., Kronawitter, S., Hannig, F., Teich, J., Lengauer, C.: Automating the development of high-performance multigrid solvers. Proc. IEEE **106**(11), 1969–1984 (2018). Special issue: From High-Level Specification to High-Performance Code
77. Schmitt, C., Schmid, M., Kuckuk, S., Köstler, H., Teich, J., Hannig, F.: Reconfigurable hardware generation of multigrid solvers with conjugate gradient coarse-grid solution. Parallel Process. Lett. **28**(4), 20 pp. (2018). Article 1850013
78. Schmitt, J., Kuckuk, S., Köstler, H.: Optimizing Geometric Multigrid Methods with Evolutionary Computation (2019). arXiv:1910.02749
79. Schweikl, S.: Multigrid for the SPIRAL prototype in Scala. Bachelor's thesis, University of Passau (2017)
80. Siegmund, N., Grebhahn, A., Apel, S., Kästner, C.: Performance-influence models for highly configurable systems. In: Proceedings of the European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 284–294. ACM Press, New York (2015)
81. Spampinato, D., Fabregat-Traver, D., Bientinesi, P., Püschel, M.: Program generation for small-scale linear algebra applications. In: Proceedings of the International Symposium on Code Generation and Optimization (CGO), pp. 327–339. ACM, New York (2018)
82. Tang, Y., Chowdhury, R.A., Kuszmaul, B.C., Luk, C.K., Leiserson, C.E.: The Pochoir stencil compiler. In: Proceedings of the 23rd Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 117–128. ACM, New York (2011)
83. Treibig, J., Hager, G., Wellein, G.: LIKWID: a lightweight performance-oriented tool suite for $\times$86 multicore environments. In: Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW), vol. 1, pp. 207–216 (2010). International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)
84. Trottenberg, U., Oosterlee, C.W., Schuller, A.: Multigrid. Academic, London (2001)
85. Unat, D., Cai, X., Baden, S.B.: Mint: realizing CUDA performance in 3D stencil methods with annotated C. In: Proceedings of the International Conference on Supercomputing (ICS), pp. 214–224. ACM, New York (2011)
86. van Rossum, G.: Python Reference Manual. Tech. Rep. CS-R9525, Centrum voor Wiskunde en Informatica (CWI) (1995)

www.dbooks.org

87. Vanka, S.: Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. J. Comput. Phys. **65**(1), 138–158 (1986)
88. Wienands, R., Joppich, W.: Practical Fourier Analysis For Multigrid Methods. Chapman Hall/CRC Press, Boca Raton (2005)
89. Williams, S., Watermann, A., Patterson, D.: Roofline: an insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009)
90. Zhang, N., Driscoll, M., Markley, C., Williams, S., Basu, P., Fox, A.: Snowflake: a lightweight portable stencil DSL. In: Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 795–804. IEEE, New York (2017)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



ExtraPeak: Advanced Automatic Performance Modeling for HPC Applications



Alexandru Calotoiu, Marcin Copik, Torsten Hoefler, Marcus Ritter, Sergei Shudler, and Felix Wolf

Abstract Performance models are powerful tools allowing developers to understand the behavior of their applications, and empower them to address performance issues already during the design or prototyping phase. Unfortunately, the difficulties of creating such models manually and the effort involved render performance modeling a topic limited to a relatively small community of experts. This article summarizes the results of the two projects Catwalk, which aimed to create tools that automate key activities of the performance modeling process, and ExtraPeak, which built upon the results of Catwalk and worked toward making this powerful methodology more flexible, streamlined and easy to use. The sew projects both provide accessible tools and methods that bring performance modeling to a wider audience of HPC application developers. Since its outcome represents the final state of the two projects, we expand to a greater extent on the results of ExtraPeak.

1 Introduction

High-performance computing (HPC) is a key technology of the twenty-first century. Numerous application examples, ranging from the improved understanding of matter to the discovery of new materials and from the study of biological processes to the analysis of social networks, give evidence of its tremendous potential. Mastery of this technology will decide not only on the economic competitiveness of a society but will ultimately influence everything that depends on it, including the society's welfare and stability. Moreover, there is broad consensus that high-performance

A. Calotoiu · M. Ritter · S. Shudler · F. Wolf (✉)
Technische Universität Darmstadt, Darmstadt, Germany
e-mail: calotoiu@cs.tu-darmstadt.de; ritter@cs.tu-darmstadt.de; sshudler@anl.gov; wolf@cs.tu-darmstadt.de

M. Copik · T. Hoefler
ETH Zürich, Zürich, Switzerland
e-mail: marcin.copik@inf.ethz.ch; htor@inf.ethz.ch

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_15

www.dbooks.org

computing is indispensable to address major global challenges of humankind such as climate change and energy consumption. However, the demand for computing power needed to solve problems of such enormous complexity is almost insatiable. In their effort to answer this demand, supercomputer vendors work alongside computing centers to find good compromises between technical requirements, tight procurement and energy budgets, and market forces that dictate the prices of key components. The results are sophisticated architectures that combine unprecedented numbers of processor cores into a single coherent system, leveraging commodity parts or at least their designs to lower the costs where in agreement with design objectives.

Exploiting the full power of HPC systems has always been hard and is becoming even harder as the complexity and size of systems and applications continues to grow. On the other hand, already today the savings potential in terms of energy and CPU hours that application optimization can achieve is enormous [5]. As the number of available cores increases at tremendous speed, reaping this potential is becoming an economic and scientific obligation. For example, an exascale system with a power consumption of 20 MW (very optimistic estimate) and 5000 h of operation per year would—assuming an energy price of 0.1€ per kWh—produce an energy bill of 10 M€ per year.

Ever-growing application complexity across all domains, including but not limited to theoretical physics, fluid dynamics, or climate research, requires a continuous focus on performance to productively use the large-scale machines that are being procured. However, designing such large applications is a complex task demanding foresight since they require large time investments in development and verification and are therefore meant to be used for decades. Thus, it is important that the applications be efficient and potential bottlenecks are identified early in their design as well as throughout their whole life cycle. Continuous performance analysis starting in early stages of the development process is therefore an indispensable prerequisite to ensure early and sustained productivity.

Tuning an application means finding the sweet spot in its combined design and configuration space. Unfortunately, the sheer size of this space renders its exhaustive traversal via performance experiments prohibitive. In the absence of alternatives, many developers still rely on experiments, trying only a small and not necessarily representative subset of the available design and configuration options. Because of their limited view, they more than often overlook valuable optimization opportunities or miss latent performance limitations whose underlying trend they did not capture.

Performance models, in contrast, allow the design space to be explored much faster and much more thoroughly. Although often based on simplifying assumptions, they offer tremendous insight at the small cost of evaluating a formula. A model can be easily used to balance important trade-offs and adjust design parameters such that close to optimal performance is achieved. Such models allow problems in applications to be detected early on, long before they manifest themselves in production runs, and their severity to be determined when the cost of eliminating the problems is comparably small. If the problem is discovered later,

dependencies between its source and the rest of the code that have grown over time can make remediation much harder. However, finding performance models is both hard and time consuming, which is why many developers shy away from it. Sometimes such models are simply built on inaccurate back-of-the-envelope calculations, rough estimates, simple and manual spreadsheet calculations, or even only developer intuition, which may be misleading.

An analytical performance model expresses the performance of an application in terms of a purely analytical expression [20, 27, 31]. A target metric m (e.g., execution time, energy, or number of floating-point operations) is represented as a function $m = f(x_1, \ldots, x_n)$ of one or more parameters x_i (e.g., the number of cores or the size of the input problem). To make statements about application performance that can be relied on under changing conditions, it is usually not enough to focus on any single parameter in isolation. The effect that one varying parameter has on performance must be considered in the context of the variation of other relevant parameters, including algorithm options, tuning parameters such as tiling, or characteristics of the input data set. However, often it is not obvious which parameters are truly performance-relevant and should be included. In general, the decision whether to include a certain parameter or not has to trade off different criteria. Models with fewer parameters are easier to generate and maintain and provide more high-level insight, whereas models with more parameters can be more accurate because they consider more effects. Abstract application performance models with a reasonably small number of parameters can be designed and maintained by application developers while a system model can only be provided by system experts. Simpler models can be used as an interface to application developers and algorithm designers, while more complex models can be used for detailed tuning and projections. The task of modeling the performance of an application is rather complex and time-consuming though. This is why—in spite of its potential—it is rarely used in practice. However, with the help of automatic tools that support the creation of accurate performance models, this powerful methodology could spread across a much wider audience of HPC application developers.

This article summarizes the results of the Catwalk and ExtraPeak projects, which set out to improve this situation. The main goal of Catwalk, the first of the two projects, was to make performance modeling more attractive for a broader audience of HPC application developers by providing a method to create insightful performance models as automatically as possible, in a simple and intuitive way. Given the success of Catwalk, the follow-up project ExtraPeak aimed to improve upon the basic performance modeling method by allowing the models to include more than one parameter, while preserving the speed and accuracy of the original model generation process. Since ExtraPeak represents the more advanced state of our research and the outcome of Catwalk is already summarized elsewhere [47], we take the liberty of expanding mostly on the achievements of ExtraPeak. The two projects are part of a wider pioneering effort to construct performance models automatically in multiple application areas also beyond HPC, ranging from enterprise systems [7] to databases [15] and software product lines [40].

www.dbooks.org

2 Overview of Contributions

Although we focus on the contributions of ExtraPeak, we start by offering a short overview of Catwalk and its achievements, given that the former builds upon the results of the latter. Figure 1 summarizes the accomplishments of both projects and their relationship to each other. Results from Catwalk are shown in hatched boxes while results from ExtraPeak are shown in solid boxes.

2.1 Catwalk

The most important goal of Catwalk was to provide an automated method for constructing performance models. The focus was on the discovery of *s*calability bugs, achieved by creating models describing the performance of different parts of a program when scaled to larger processor configurations and demonstrated using several realistic applications, including Sweep3D, MILC, and HOMME [10]. The method is the foundation of the performance modeling tool Extra-P, a major outcome of Catwalk, which has been released under an open-source license. Extra-P enabled numerous application case studies to showcase the type of insights this analysis can provide, including UG4 [44], an unstructured-grid package, as

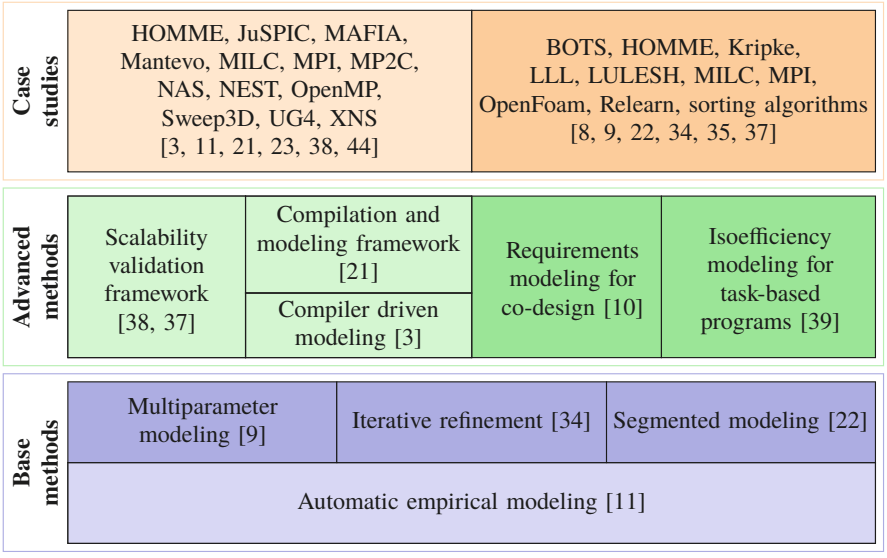


Fig. 1 Main contributions overview. Contributions in boxes with darker shades and solid fill represent work completed in the ExtraPeak project, whereas contributions in boxes with lighter shade and hatched fill represent contribution from the Catwalk project

well as several state-of-the-art MPI [37] and OpenMP implementations [24]. In Catwalk, we also worked on approaches leveraging the performance-modeling base method to allow new research avenues to be explored. The results include a scalability test framework that combines performance modeling with expectations to systematically validate the scalability of libraries [37], a tool that automatically instruments applications to generate performance models at runtime [4], and a fully static method to derive numbers of loop iterations from loops with affine loop guards and transfer functions that can be used to limit the performance model search space [22].

2.2 ExtraPeak

In ExtraPeak, we improved the basic modeling approach and expanded it to allow the analysis of multiple parameters simultaneously, automating the search space generation for models and even handling discontinuities in the model space. A continuous goal in the development of our methods was to ensure flexibility and ease of use. Our main accomplishments can be divided into three categories: new features for and improvements of the base method, advanced methods building upon it to explore new ways of expressing and understanding performance, and finally case studies leveraging Extra-P to gain insights into the performance of specific applications.

Base Methods The extensions of the base method cover both multiple model parameters and a refined model search for a single parameter:

- We extended the basic approach to allow insightful modeling of any combination of application execution parameters while using heuristics to decrease the time to find the best models quickly without compromising their quality [11].
- We developed an algorithm to detect segmentation in a sequence of performance measurements and estimate the point where the behavior changes, allowing complex irregular behaviors to be modeled [23].
- We designed a new model-generation algorithm by which the search space is built and automatically refined on demand relieving the user from the burden of search space selection [34].

Advanced Methods The flexibility of Extra-P together with the width of models it can express and the insight they offer make it easy to tailor it to advanced tasks related to the exploration of multi-dimensional performance spaces:

- Task-based programming offers an elegant way to express units of computation and the dependencies among them, making it easier to distribute the computational load evenly across multiple cores. We introduced an automated empirical method for finding the isoefficiency function of a task-based program, binding efficiency, core count, and the input size in one analytical expression. The insights

www.dbooks.org

gained via these expressions can be used to co-design programs and shared system resources [38].
- In the co-design process, a fundamental aspect of the application requirements are the rates at which the demands for different resources grow as a code is scaled to a larger machine. We showed how automated performance modeling can be used to quickly predict application requirements for varying scales and problem sizes [12].

Case Studies In ExtraPeak, we continued the series of case studies started in Catwalk, in which we confirm prior expectations or discover the existence of previously unknown scalability bottlenecks—often in collaboration with other research teams. Many of them were conducted when we extended our base methods or developed advanced methods and are described in the work cited above. In addition, we conducted further case studies with the sole aim of better understanding the performance of their application targets:

- We helped validate the complexity of Relearn, a code that simulates structural plasticity in the brain. Inspired by hierarchical methods for solving n-body problems in particle physics, Relearn uses a scalable approximation algorithm with the complexity $O(n \cdot \log n)$, which can simulate the structural plasticity of up to 10^9 neurons—four orders of magnitude more than the naïve $O(n^2)$ version previously available [35].
- We investigated two implementations of the LLL lattice basis reduction algorithm in the popular NTL and fplll libraries, which helps to assess the security of lattice-based cryptographic schemes, to validate their complexity in practical usage scenarios. This task reverses the perspective of classic HPC applications. In the field of cryptography, high algorithmic complexity is a desirable trait that characterizes the hardness of breaking certain security protocols [9].
- We performed an analysis of parallel sorting algorithms and further MPI implementations using the framework for continuous scalability validation previously developed during the Catwalk project [39].

In the remainder of the article, we further describe the extensions of the base method as well as the advanced methods building upon it. Since this article focuses on advances in automatic performance modeling, we cover only case studies conducted alongside these developments, but do not include more details on the remaining ones listed above. For those, we refer the reader to the cited literature. The overview of our technological contributions is followed by a summary of ongoing developments in our project, a review of related approaches, and eventually a brief conclusion. However, because it is fundamental to everything we did in ExtraPeak, we first provide a short introduction to Extra-P, the tool whose development was started in Catwalk.

3 Extra-P

The key result of Catwalk has been a method to identify *scalability bugs*. A scalability bug is a part of the program whose scaling behavior is unintentionally poor, that is, much worse than expected. As computing hardware moves towards exascale, developers need early feedback on the scalability of their software design so that they can adapt it to the requirements of larger problem and machine sizes. In addition to searching for performance bugs, the models our tool produces also support projections that can be helpful when applying for the compute time needed to solve the next larger class of problems. For a detailed description, the reader may refer to Calotoiu et al. [10].

The usual input of our tool when used as a scalability bug detector is a set of performance measurements on different processor counts $\{p_1, \ldots, p_{max}\}$ in the form of parallel profiles. As a rule of thumb, we use five or six different configurations. The output of our tool is a list of program regions, ranked by their predicted execution time at a chosen target scale or by their asymptotic execution time. We call these regions *kernels* because they define the code granularity at which we generate our models.

Model Generation When generating performance models, we exploit the observation that they are usually composed of a finite number n of predefined terms, involving powers and logarithms of p:

$$f(p) = \sum_{k=1}^{n} c_k \cdot p^{i_k} \cdot log_2^{j_k}(p). \tag{1}$$

This representation is, of course, not exhaustive, but works in most practical scenarios since it is a consequence of how most computer algorithms are designed. We call it the *performance model normal form* (PMNF). Moreover, our experience suggests that neither the sets $I, J \subset \mathbb{Q}$ from which the exponents i_k and j_k are chosen nor the number of terms n have to be arbitrarily large or random to achieve a good fit. Thus, instead of deriving the models through reasoning, we only need to make reasonable choices for n, I, and J and then simply try all assignment options one by one. For example, a default we often use is $n = 3$, $I = \left\{\frac{0}{2}, \frac{1}{2}, \frac{2}{2}, \frac{3}{2}, \frac{4}{2}, \frac{5}{2}, \frac{6}{2}\right\}$, and $J = \{0, 1, 2\}$. A possible assignment of all i_k and j_k in a PMNF expression is called a *model hypothesis*. Trying all hypotheses one by one means that for each of them we find coefficients c_k with optimal fit. Then we apply cross-validation to select the hypothesis with the best fit across all candidates. As an alternative to the number of processes p, our method can also support other model parameters such as the size of the input problem or other algorithmic parameters—as long as we vary only one parameter at a time.

www.dbooks.org

Our tool models only behaviors found in the training data. We provide direct feedback information regarding the number of runs required to ensure statistical significance of the modeling process itself, but there is no automatic way of determining at what scale particular behaviors start manifesting themselves. This version of our method is most effective for regular problems with repetitive behavior, whereas irregular problems with strong and potentially non-deterministic dynamic effects require enhancements which we detail in Sect. 4.2.

Open Source Release The Extra-P performance-modeling tool has been made available online under an open-source license.[1] Users have access not only to the software but also to documentation material describing both our method and its implementation.[2] We have been actively supporting the use of Extra-P at several organizations, among them the High Performance Computing Center Stuttgart, TU Darmstadt, Lawrence Livermore National Laboratory, FZ Jülich, and the University of Washington, just to name a few.

Figure 2 shows how the results of the model generator can be interactively explored. The GUI annotates each call path with a performance model. The formula represents a previously selected metric as a function of the number of processes, and allows other parameters to be represented as well. The user can select one or more call paths and plot their models on the right. In this way, the user can visually compare the scalability of different application kernels.

The profiles needed as input for the model generator are created in a series of performance experiments. To relieve the user from the burden of manually submitting large numbers of jobs and collating their results, we use the Jülich Benchmark Environment (JUBE) [26], a workflow manager developed at Forschungszentrum Jülich.

The tool was presented at multiple tutorials at conferences such as EuroMPI and Supercomputing, as well as at numerous VI-HPS and HKHLR tuning workshops. Following a 90-min theoretical explanation of the method and the tool, users were able to model the performance of two example applications, SWEEP3D and BLAST, in a 90-min practical session. Using previously prepared measurement data, they were able to generate models for the entire codes, evaluate the results, and understand the scaling behavior of the two applications. With this knowledge, attendees are able to apply Extra-P to their own applications, once the required performance measurements have been gathered. Because Extra-P is compatible with Score-P an established infrastructure for performance profiling, even collecting these measurements is straightforward.

[1] http://www.scalasca.org/software/extra-p/download.html.

[2] http://www.scalasca.org/software/extra-p/documentation.html.

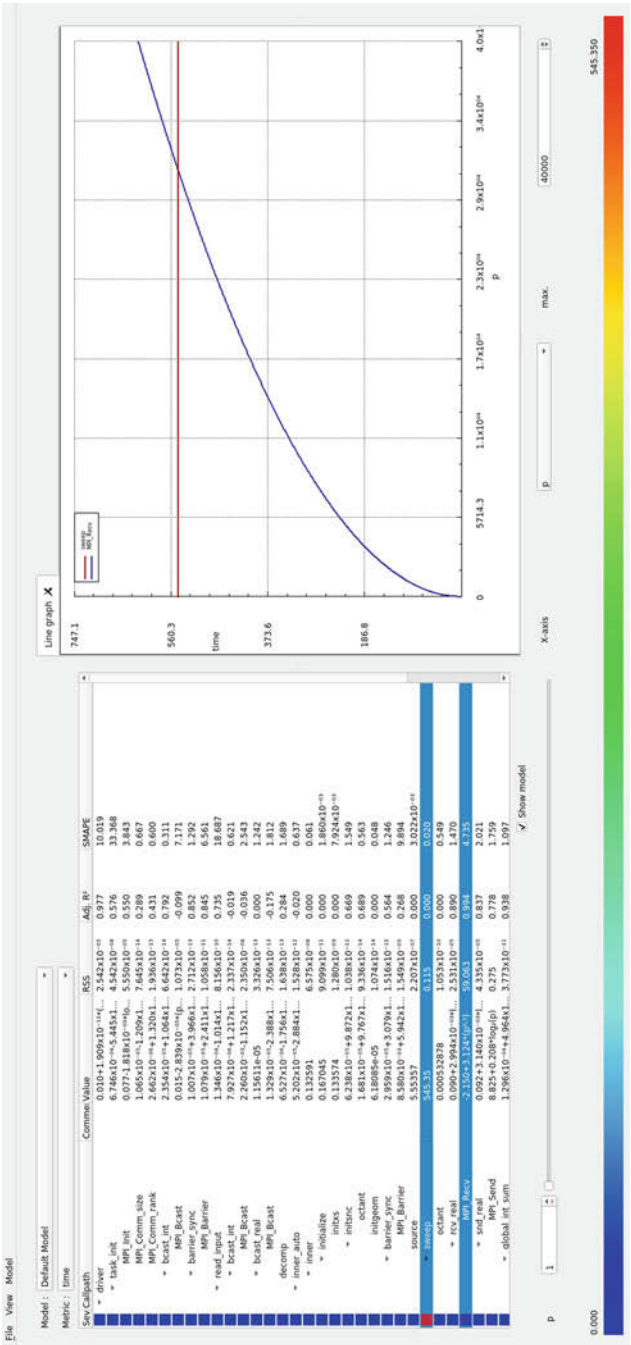


Fig. 2 Interactive exploration of performance models. The screen shot shows performance models generated for call paths in SWEEP3D, a neutron transport simulation

www.dbooks.org

4 Developments of the Base Methods

While the tool created during the Catwalk project is already widely used in the HPC community and provides meaningful insights to developers seeking to understand the performance of their codes, we have added significant new features during the ExtraPeak project which we detail below, most important among them the ability to model multiple parameters simultaneously.

4.1 Multi-Parameter Modeling

Common questions asked by developers when trying to understand the behavior of applications are:

- How does application performance change when more processors are used?
- How does application performance change when the problem size is increased or decreased?

When considering the pressure on applications to judiciously use computing resources both questions must be answered, and a new vital question arises:

- Are the effects of processor variation and problem-size variation independent of each other or can they amplify each other?

For example, a weak-scaling run of the kernel `SweepSolver` in Kripke [28], a particle transport proxy application, has a runtime model for processor variation of $t(p) = O(p^{1/3})$ and a runtime model for varying the number of dimensions of $t(d) = O(d)$. The number of dimensions influences the problem size proportionally. It now needs to be determined how these two factors play together. Depending on their interaction, the application is scalable or not. For example, it would make a huge difference whether the combined effect of processor variation and number of dimensions was $t(p, d) = O(p^{1/3} \cdot d)$ or $t(p, d) = O(p^{1/3} + d)$.

We expanded the original performance model normal form presented in Sect. 3 to include multiple parameters.

$$f(x_1, \ldots, x_m) = \sum_{k=1}^{n} c_k \cdot \prod_{l=1}^{m} x_l^{i_{kl}} \cdot log_2^{j_{kl}}(x_l) \tag{2}$$

This expanded normal form allows a number m of parameters to be combined in each of the n terms that are summed up to form the model. Each term allows each parameter x_l to be represented through a combination of monomials and logarithms. The sets $I, J \subset \mathbb{Q}$ from which the exponents i_{kl} and j_{kl}, respectively, are chosen can be defined as in the one-parameter case.

Of course, if multiple parameters are considered, performance experiments have to be conducted for all combinations of parameter values and the total number of

experiments that is required grows accordingly. While this might be manageable if the number of parameters considered is small enough and/or the cost of an individual experiment is very small, another and more serious problem emerges even for two and three parameters, namely the combinatorial explosion of the model search space.

Therefore, multi-parameter modeling was outside the reach of automatic methods due to the exponential growth of the model search space. We developed a new technique to traverse the search space rapidly and generate insightful performance models that enable a wide range of uses from performance predictions for balanced machine design to performance tuning. The details can be found in the work of Calotoiu et al. [11], but we present the most important heuristics here, together with a summary of the evaluation.

Hierarchical Search The idea is to first obtain single parameter models for each individual parameter. Once we have these models, all that is left is to compare all additive and multiplicative options of combining said models into one multi-parameter model and to choose the one with the best fit.

The size of the search space for this approach is as follows, given m parameters and one n-term model for each of them. We must combine all subsets of terms of each single-parameter model with each subset of terms of each other single parameter model. The number of subsets of a set of n elements is 2^n, so the total size of the search space is $2^{n \cdot m}$.

Assuming there are three parameters, the single-parameter models for all of them have been computed and each model has three terms (the worst case scenario for search space cardinality in this case), the number of hypotheses that have to be tested is $2^{3*3} = 512$. Adding the 3 times 25 steps needed to generate the single-parameter models, we will need to look at most at 587 models to find the best fit, compared to the $6.51 \cdot 10^{14}$ in the unoptimized approach.

Evaluation To evaluate the multi-parameter modeling approach we quantify the speedup of the model search in comparison to an exhaustive traversal of the same search space. Furthermore, we determine the frequency at which our heuristics lead to models that differ from the ones the exhaustive search produces. In those cases where the models we discover are different, we analyze these differences and discuss their impact on the quality of the results. Because traversing the entire search space for three or more parameters is prohibitively time consuming even with a very small number of potential terms, we allow only at most two model parameters for the purpose of this comparison.

The evaluation is divided into two parts. First, we examine how closely the models generated both through exhaustive search and with the help of heuristics resemble inputs derived from synthetically generated functions. This allows our results to be compared with a known optimal model. Second, we compare the results of both approaches, when applied to actual performance measurements of scientific codes, which factors in the effects of run-to-run variation.

www.dbooks.org

Synthetic Data We generated 100,000 test functions by instantiating our normal form from Eq. 1 with random coefficients. Our model generator responded in three different ways:

1. Optimal models. The most common result (ca. 95%) is that the heuristically determined model, the model determined through an exhaustive search, and the known optimal model are identical.
2. Lead-order term and its coefficient identified, smaller term not modeled by either method. In rare cases, neither modeling approach is capable of detecting the smaller term and they both only model the lead-order term. The effect on the quality of the resulting models is very small, and an attempt to model such small influences will often lead to noise being modeled instead.
3. Lead-order term and its coefficient identified, smaller additive term only modeled by exhaustive search. In this case the heuristic approach fails to identify the parameter with a much smaller effect. The effect on the quality of the resulting model is again negligible.

Table 1 displays the number of times the modeling identified the entire function correctly and the times only the lead-order term was identified correctly. The lead-order term was correctly identified in all test cases. The difference in time required to obtain the 100,000 models is significant: 1.5 h when using the heuristics compared to 107 h when trying out all models.

Application Measurements In addition to synthetic data, we evaluated our heuristics with three scientific applications: Kripke, Cloverleaf, and BLAST. For BLAST we used two qualitatively different solvers and will therefore present separate results. Real data sets come with new challenges, such as not knowing the optimal model, and indeed no guarantees that the assumptions required for our method hold, namely that the optimal model is described by one and only one function and that the function is part of the search space. Figure 3 shows the results of both applying the heuristics and searching the entire solution space. As expected, in the overwhelming majority of cases the two approaches provide the same result (84%), or at least present the same lead-order term (14%). In about 2% of the cases the models differ. The reason is that noise and outliers occurring in real data sets are not limited to any arbitrary threshold compared to the effect of different parameters on performance. The projection used by the heuristics to generate single-parameter

Table 1 Evaluation of heuristics using synthetic functions

Search type	Heuristic	Exhaustive
Optimal models identified	95,480 [95.5%]	96,120 [96.1%]
Lead-order term identified (including coefficient)	4,520 [4.5%]	3880 [3.9%]
Lead-order term not identified	0 [0%]	0 [0%]
Modeling time	1.5 hrs.	107 hrs.

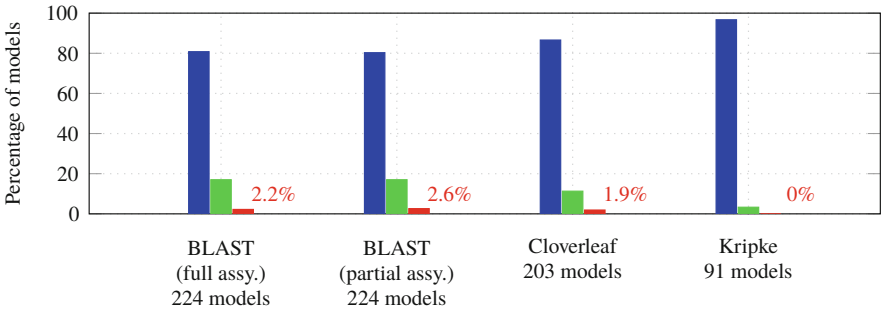


Fig. 3 Comparison of performance models obtained for all kernels of scientific applications using either our heuristics or a full traversal of the search space. For each application, we show the percentage of times where the resulting models were identical (left bar), where only the lead-order terms and their coefficients were the same (center bar), and where the lead-order terms were also different (right bar)

models out of multi-dimensional data reduces noisy behavior to a higher degree than the exhaustive search does. Therefore, in these rare cases, the heuristic approach results in models with a slower growth rate than the ones identified through an exhaustive search. The optimal model is not necessarily the one identified by the exhaustive search, as noise could be modeled alongside the parameter effects.

In all three cases, the model generation for an entire application took only seconds and was at least a hundred times faster than the exhaustive search. Generating performance models for an entire application means one model per call path and target metric. The search space reduction in all three cases was five orders of magnitude (from 4,250,070 model hypotheses down to 66 per call path and target metric).

Discussion The evaluation with synthetic and real data demonstrates that our heuristics can offer results substantially faster than an exhaustive search—without significant drawbacks in terms of result quality. For three or more parameters, the size of the search space would have prevented such a comparison altogether, which also means that the exhaustive search presents no viable alternative beyond two parameters.

4.2 Segmented Modeling

Although our method is very powerful it struggles with the situation where performance data representing two or more different phenomena need to be combined into a single performance model, effectively being a function composed of multiple segments. This not only generates an inaccurate model for the given data, but can also either fail to point out existing scalability issues or create the appearance of such issues when none are present.

www.dbooks.org

We have developed an algorithm to help Extra-P detect segmentation in the data before models are generated. Its input is a set of performance measurements, while the output indicates whether the given measurements show segmented behavior or not. If the data turns out to be segmented, the algorithm tries to identify the change point. With this information, Extra-P can generate separate models for each segment and/or request new measurements if any segment is too small for model generation. For more information, we refer the reader to Ilyas et al. [23].

Our method correctly identified segmentation in more than 80% of 5.2 million synthetic tests and confirmed expected segmentation in three application case studies. The results of this evaluation show that the proposed algorithm can be used as an effective way to find segmentation in performance data when creating empirical performance models. The suggested algorithm does not require any extra effort on the user's side, and can work very well on as few as six points.

4.3 Iterative Search Space Refinement

While Extra-P strives to be easy to use and provide meaningful insights even to users without an extensive background in performance modeling, the version we developed in the Catwalk project still required the manual pre-configuration of the search space. This adds a layer of complexity to the modeling process, and requires the user to have an idea of what type of models can be expected from his application. Should the search space not include a model found in the code, the method will still try to approximate by selecting the one best fitting the data from the set of options available, but the result will be less accurate. Furthermore, noise in the data often leads to models that indicate a worse behavior than there actually is as sometimes a model in the search space fits not only the behavior we are trying to capture but also the noise, leading to overfitting. We have developed a new iterative model-generation approach, where we configure the search space on demand and iteratively raise the accuracy of the model until no meaningful improvement can be made. In this way, we increase both the ease of use for Extra-P and its range of application without sacrificing accuracy. For details, we refer the reader to the work by Reisert et al. [34]. In the following we show a summary of the results, showing the improvements that the new approach provides.

We used measurements from previous case studies to evaluate our new algorithm on measured data. The measurements include a variety of call paths (i.e., kernels) and different metrics, such as runtime, number of function calls, memory footprint, and network traffic.

The results of the comparison, which are presented in Table 2, show that, when the last (i.e., largest) measured data point is excluded from the data used to calculate the model, the model produced by our new algorithm allows for a better prediction of the last point in 19–65% of the cases, which corresponds to 53–85% of those models that changed in each benchmark. Although some predictions do get worse,

Table 2 Comparison of the original and our improved algorithm, using data from previous case studies, showing the quality of predictions of the last data point when that point is not used for modeling

Benchmark	Number of points	Model count	Model predictions (percentage of all models)			Mean relative prediction error [%]	
			better	same	worse	before	now
Sweep3D [11]	7	96	26.04	56.25	17.71	17.26	6.31
HOMME [11]	9	670	18.81	68.51	12.69	3.69	3.03
MILC [11]	9	1496	30.95	56.48	12.57	36.71	14.53
UG4 [44]	5	2026	52.62	38.01	9.38	68.30	15.58
MPI collect. [38]	7–8	26	65.38	7.69	26.92	52.53	15.89
BLAST [9]	5	103	31.07	41.75	27.18	34.92	10.38
Kripke [9]	5	36	36.11	38.89	25.00	33.05	8.32
Total	5–9	4453	39.12	49.11	11.77	45.71	12.97

the mean relative prediction error decreases across all applications, in all but one case even significantly.

Not shown in the table is the number of models that are constant, which has considerably increased in every single case study (from 44 to 76% overall). Because the synthetic evaluation has shown that our new algorithm is able to recognize constant functions more reliably, this indicates that the previous algorithm might have modeled noise or tried to fit a PMNF function to inaccurate measurements.

With iterative refinement we remove the need for a predefined search space and also significantly reduce the number of false positives by being more resilient to noisy measurements of constant behavior. Most of the models generated with this new algorithm are able to make predictions that are equally or even more accurate than before. We therefore open the way for a performance modeling workflow that is more automated than ever and equips developers with a tool that helps them efficiently understand the performance of their applications.

5 Developments of the Advanced Methods

Extra-P is at its core a tool to generate a human-readable function out of a set of inputs. The flexibility it offers and the features we added over time made it attractive for uses beyond identifying performance bugs in parallel applications. Some approaches, such as the compilation and modeling framework by Bhattacharyya et al. [4], were developed during the Catwalk project. The scalability framework [37] was also developed during the Catwalk project, but we have further refined and expanded it during the ExtraPeak project, and also tested a number of parallel sorting algorithms using this framework [39]. Two research directions newly investigated during the ExtraPeak project are requirements engineering and iso-efficiency modeling, and are discussed below.

www.dbooks.org

5.1 Lightweight Requirements Engineering

Co-designing applications and the system is a powerful technique to ensure early and sustained productivity as well as good system design, especially in high-performance computing where the cost of systems is very high and applications are expected to remain in use for long periods of time. In their early phases, such co-designs often rest on back-of-the-envelope (BOE) calculations. In general, such calculations allow problems in applications to be detected early on and their severity to be determined years before the machine is installed or the first prototype becomes available. This is increasingly important since mitigating such problems can often take several personyears. On the system side, BOE calculations allow designers to adjust system parameters to target applications, for example, they can be used to determine the required bytes-to-flop ratio of memory, network, or even the file system. In addition, they can be used to determine required memory sizes, usability of accelerators and co-processors, and even the number of sockets and size of shared-memory domains in the target system.

We automate these BOE calculations in a lightweight requirements analysis for scalable parallel applications. We introduce a minimal set of hardware-independent application-centric requirements that cover the most significant aspects of application behavior. Combining performance profiling [1, 8] and stack-distance sampling [3] with a lightweight automatic performance-modeling method [10, 11], we generate empirical models of these requirements that allow projections for different numbers of processes and problem sizes.

As the foundation of our approach, we define a very simple notion of requirements that supports their quantification in terms of the amount of data to be stored, processed, or transferred by an application. Knowing these numbers alone does not target a precise prediction of application runtime but can serve as an indicator of the relative importance of certain system resources and how this ratio changes as we scale a program to a larger system. Ultimately, our requirements are expressed in the form of empirical models that allow projections for different numbers of processes and problem sizes.

Application-Centric Requirements We choose requirements to be purely application centric, that is, we do not make any assumption about the hardware other than the ability to run the code as is. Hence, all our requirement metrics refer to data flow at the interface between hard- and software—not between lower layers of the hardware. While specific hardware features could improve the rate at which the requirements are fulfilled, the classes of behavior our requirement models capture will not change. For example, even if revolutionary hardware features double the speed at which floating-point computations are performed, if the number of floating-point computations that need to be performed grows quadratically with the number or processes, while all other requirements remain constant, the floating-point requirement will remain the bottleneck for that particular application as it scales up.

Table 3 Requirement metrics

Resource	Metric
Memory footprint	# Bytes used (resident memory size)
Computation	# Floating-point operations (#FLOP)
Network communication	# Bytes sent / received
Memory access	# Loads / stores; stack distance

Since it is currently the predominant programming model and also expected to be highly influential in the future, we stipulate that of each target application an MPI version exists. Application requirements are then expressed as a set of functions $r(p, n)$ that predict the demand for resource r depending on the number of processes p and the problem size per process n. Because we regard thread-level concurrency merely as a way to satisfy the requirements, we consider requirements not below the granularity of processes, which may nevertheless be multithreaded—either locally or by launching GPU kernels.

Currently, we consider the requirement metrics listed in Table 3, classified by the resource they refer to. I/O would be handled analogously to the network communication requirement. None of our analyzed applications includes significant I/O traffic, we therefore refrain from including I/O metrics in this analysis. Our metrics characterize application requirements in terms of space (i.e., memory consumption) and "data metabolism" (i.e., bytes processed in floating-point units or exchanged via memory and network). Because the amount of data moved between processor and memory subsystem alone is barely a reliable indicator of the pressure an application exerts on the memory subsystem, we also consider memory access locality.

Co-design The key point of our method is to *guide the programmer to find application bottlenecks relative to an architecture as well as to guide the architect to find system bottlenecks that a given application would experience.* Our requirements models are functions of the number of MPI processes p and the input problem size n. To compare the requirements of an application on two different architectures, all we need to do is to calculate the application requirements using the values for p and n the application would use on these two systems. For details regarding the collection of requirement metrics and more detailed case-studies we refer the readers to Calotoiu et al. [12]. In the following we wish to present a brief example for the types of insights requirements modeling offers.

LULESH is a widely studied proxy application in DOE co-design efforts for exascale which calculates simplified 3D Lagrangian hydrodynamics on an unstructured mesh. The problem size per process is defined as the simulated volume per process. The growth rates of all requirements with respect to both problem size and process count are very close to ideal. With the current implementation, the multiplicative effect process count and problem size per process have on computation and communication for LULESH is a small obstacle in tailoring and scaling the application to run on different systems. The growth rates are slow enough to limit these issues at anything except the most extreme scales. Having introduced

www.dbooks.org

Table 4 Per-process requirements models

	Metric	Model
LULESH	#Bytes used	$10^5 \cdot n \log n$
	#FLOP	$10^5 \cdot n \log n \cdot p^{0.25} \log p$ ⚠
	#Bytes sent & received	$10^3 \cdot n \cdot p^{0.25} \log p$ ⚠
	#Loads & stores	$10^5 \cdot n \log n \cdot \log p$
	Stack distance	Constant

the per process requirement models, we can now showcase the workflow to evaluate a possible system upgrade taking account of these requirements. Let us consider the scenario where LULESH is working on a given system, but needs to be deployed to a larger system of the same type, for example one having twice the number or racks.

The requirements of LULESH are listed at the top of the table as part of Step I. Following this process, we can now draw conclusions regarding system utilization, requirements balance, and usefulness of a particular upgrade. The ratios between new and old problem sizes indicate how the largest problem size that can be solved changes, both per process and overall. The ratios between new and old requirements indicate which system components will experience an increased load relative to other components (Table 4).

The requirements of LULESH can be expressed as the product of single-parameter functions that either depend the problem size per process or the number of processes. When doubling the racks, only the value of p changes, and in this particular case, all terms depending on n can be reduced when determining the ratios of the changing requirements. This means that these ratios are valid regardless of the problem size per process. This will not generally be true as it depends on the specific relative upgrade. That the number of processes affects computation and communication means that these requirements increase slightly. Luckily, computation and communication only increase by 20% and will therefore allow LULESH to solve an overall problem twice as large with only a small performance degradation.

Discussion The workflow we propose leverages these models to enable system designers and application developers to ponder various upgrade and design options. We characterize performance in terms of relative requirement changes—from one system or one application to another. This pattern indeed matches the common case, where an initial version of an application running on an initial system already exists. And even if no such system exists, our approach can successfully help compare design options. The main advantage of our approach in relation to architecture-specific performance models, which are traditionally hard and laborious to produce with high accuracy, however, is the small effort on the one hand and the low complexity of the models on the other, facilitating quick insights at low cost—easily at the scale of an entire compute-center workload (Table 5).

Table 5 Workflow for determining the requirements of LULESH after doubling the number of racks

I: Create requirement models for memory footprint, communication, computation, and memory access.

Metric	Process scaling and problem scaling
#FLOP	$n \log n \cdot p^{0.25} \log p$
#Bytes sent & recv.	$n \cdot p^{0.25} \log p$
#Loads & stores	$n \log n \cdot \log p$
#Bytes used	$n \log n$

II: Determine the new maximum number of processes and new memory available per process that the upgraded system supports.

Configuration parameter	Old	New
Processes count	p	$p' = 2p$
Memory	m	$m' = m$

III: Determine the new memory footprint requirement per process if all processors are used.

Metric	Old	New
#Bytes used	$n \log n$	$n' \log n'$

IV: Determine the new problem size per process such that the memory footprint equals the memory available to each process and compute the new overall problem size.

Metric	Old	New	Ratio
Problem size per proc.	$n \log n = n' \log n' = m$		1
Overall problem size	$p \cdot n$	$p' \cdot n'$	2

V: Determine the new requirements for computation, communication, and memory access.

Metric	Old	New	Ratio
#FLOP	$p^{0.25} \log p$	$(2p)^{0.25} \log 2p$	≈ 1.2
#Bytes sent & recv.	$p^{0.25} \log p$	$(2p)^{0.25} \log 2p$	≈ 1.2
#Loads & stores	$\log p$	$\log 2p$	≈ 1

www.dbooks.org

5.2 *Configuring and Understanding the Performance of Task-Based Applications*

Task-based programming models, such as Cilk [6] or OpenMP [32], are well known and as the number of cores per node continues to increase, they gain more and more attention. One major advantage of task-based programming is that it allows parallelism to be expressed in terms of tasks, which are units of computation that can be either independent, dependent on a previous task, or a prerequisite to a subsequent task. Explicitly expressing parts of the code as tasks allows the compiler to take care of all the thread management intricacies, thereby sparing the user from tedious low-level details.

However, ensuring that that the problem to be solved is large enough to require a certain number of tasks is a difficult problem, and requires extensive analysis. The efficiency of the program will decrease as more processing elements are added. The only way to ensure that efficiency remains constant, as the number of cores increases, is to increase the input size as well. This concept is embodied in the isoefficiency relation [17], which binds the number of processing elements (PEs) the application uses to the input size. It specifies by which factor the input size has to increase, with respect to the increase in the number of PEs, to maintain constant efficiency. Isoefficiency can be generalized to a two-parameter efficiency function that provides efficiency values as a function of both the PE count and the input size.

Although isoefficiency analysis is useful in understanding the scalability behavior of algorithms, it is not straightforward to apply and requires deep knowledge of the algorithm. In practice, however, task-based algorithms experience hardware limitations in the form of resource contention in general and memory contention in particular. Resources such as cache and memory controllers are limited and can negatively impact application scalability [46]. These might render theoretical isoefficiency functions not accurate enough to be used in practice. To be able to make informed decisions as to how big the input size should be in order to use all of the allocated cores efficiently, the user not only has to have a realistic isoefficiency model but also needs to understand the severity of resource contention at higher scales.

We proposed a novel practical method to automatically model the empirical efficiency functions of task-based applications [38]. Modeling the efficiency function allows us to easily derive an isoefficiency relation for any realistic target efficiency, and a carefully designed framework allows replays with different contention assumptions.

In our approach we identified three different efficiency functions for a task-based application:

1. $E_{ac}(p, n)$: The actual efficiency function of the application, modeled after the empirical results of runtime benchmarks. In this case the application runs as it is and experiences contention. Therefore, this function reflects realistic application performance including resource contention and scheduling overhead.

2. $E_{cf}(p,n)$: The contention-free efficiency function, modeled after the results of replaying empty task skeletons according to the application's task dependency graphs. The replay uses the same task dependency graphs and scheduling policy as in the original runs that were benchmarked to produce $E_{ac}(p,n)$. Since the replay is free of resource contention, this efficiency function reflects an ideal situation in which the application does not experience resource contention caused by threads accessing the same resource simultaneously.
3. $E_{ub}(p,n)$: An upper bound on the efficiency of the application. Since efficiency is defined as $\frac{S_p(n)}{p}$, an upper bound on the speedup also limits the efficiency. Considering the average parallelism $\pi(n)$ for a problem size n, we determined that $S_p(n) \leq \min\{p, \pi(n)\}$, thus we define $E_{ub}(p,n) = \min\{1, \frac{\pi(n)}{p}\}$. This function describes an ideal situation of maximum speedup that is hardly achievable in practice.

Beyond simply uncovering fundamental scalability limitations in an algorithm, we can provide insights into the impact of resource contention and determine what input size is required for any given degree of parallelism to reach a given efficiency. Further questions that our approach can answer are: What is the required core count for a given input size such that we maintain a constant, given efficiency? Which efficiency can we expect for a given number of cores and input size? Both questions are related to the co-design process when hardware designers have to understand how to make future systems suitable for both existing and future applications. Details regarding the approach and the framework required can be found in the work by Shudler et al. [38].

For an idea of the type of insights we provide, we show the summary of results for a number of task-based benchmark applications. Table 6 presents the efficiency models of the evaluated applications. There are 3 rows for each application listing the three efficiency models that we created (i.e., $E_{ub}(p,n)$, $E_{ac}(p,n)$, and $E_{cf}(p,n)$). In all the models the logarithms are binary. The *rRMSE* column is the relative root-mean-square error. It is a standard statistical factor that measures the relative differences between the observed data and the model, and is defined as: $\text{rRMSE} = \sigma/\bar{y}$, where: $\sigma = \sqrt{\sum_{i=1}^{n}(f(x_i) - y_i)^2/n}$, y_i are observed data, and $\bar{y}$ is the mean of the y_i values. The last column shows the input size n, derived from our models by letting the efficiency E be 0.8 and the core count p be 60.

All of the $E_{ac}(p,n)$ and $E_{cf}(p,n)$ models follow the same pattern $C - A \cdot f(p) + B \cdot f(p)g(n)$ that empirically emerged from our measurements. The interpretation of this pattern is that the first term, the constant C, is approximately 1 and it denotes the maximum attainable efficiency. The second term, $-A \cdot f(p)$, reflects the reduction in efficiency that occurs when we increase the core count. The last term, $B \cdot f(p)g(n)$, denotes the efficiency that we gain when we increase the input size. Together these terms reflect the interplay between the core count and the input size, and the effect it has on the efficiency. In the case of FFT, the constant B in the last term of $E_{ac}(p,n)$ is very small, which means that resource contention is a very significant factor and even large increases of the input size are not enough to offset the drop in the efficiency.

www.dbooks.org

Table 6 Efficiency models of the evaluated applications

Application	Model		rRMSE	Input size for $p = 60$
Cholesky	E_{ac}	$1.09 - 0.51 \cdot \sqrt{p} + 3.11 \cdot 10^{-2} \cdot \sqrt{p} \log n$	9.7%	$37{,}718 \times 37{,}718$
	E_{cf}	$1.14 - 0.54 \cdot \sqrt{p} + 3.4 \cdot 10^{-2} \cdot \sqrt{p} \log n$	7.8%	$24{,}685 \times 24{,}685$
	E_{ub}	$\min\{1, (2.29 + 2.35 \cdot 10^{-3} \cdot n) \cdot p^{-1}\}$	2.4%	$19{,}500 \times 19{,}500$
FFT	E_{ac}	$0.96 - 0.1 \cdot \log p + 5.08 \cdot 10^{-22} n^{4.5} \log p$	19.5%	$30{,}310 \times 30{,}310$
	E_{cf}	$1.03 - 0.16 \cdot p^{0.67} + 1.04 \cdot 10^{-2} \cdot p^{0.67} \log n$	4.8%	$15{,}800 \times 15{,}800$
	E_{ub}	$\min\{1, (1.19 \cdot 10^{-2} \cdot n^{0.67} \log n) \cdot p^{-1}\}$	4.1%	$5{,}800 \times 5{,}800$
Fibonacci	E_{ac}	$0.98 - 5.11 \cdot 10^{-3} \cdot p^{1.25} + 1.76 \cdot 10^{-3} \cdot p^{1.25} \log n$	3.5%	51
	E_{cf}	$0.97 - 1.46 \cdot 10^{-2} \cdot p^{1.25} + 9.26 \cdot 10^{-3} \cdot p^{1.25} \log n$	3.0%	51
	E_{ub}	$\min\{1, (25.48 + 0.49 \cdot n^{2.75} \log n) \cdot p^{-1}\}$	1.5%	49
NQueens	E_{ac}	$1.04 - 0.66 \cdot \sqrt{p} + 0.17 \cdot \sqrt{p} \log n$	13%	14
	E_{cf}	$1.0 - 6.21 \cdot 10^{-2} \cdot p + 1.61 \cdot 10^{-2} \cdot p \log n$	3%	13
	E_{ub}	$\min\{1, (2.18 \cdot n^{2.875} \log n) \cdot p^{-1}\}$	6.6%	12
Sort	E_{ac}	$0.99 - 9.2 \cdot 10^{-3} \cdot p^{1.5} + 2.29 \cdot 10^{-4} \cdot p^{1.5} \log n$	1.9%	350G
	E_{cf}	$1.0 - 4.61 \cdot 10^{-2} \cdot p^{0.75} + 1.62 \cdot 10^{-3} \cdot p^{0.75} \log n$	5.7%	6.6M
	E_{ub}	$\min\{1, (3.53 + 3.32 \cdot 10^{-2} \cdot \sqrt{n}) \cdot p^{-1}\}$	6.7%	1.7M
SparseLU	E_{ac}	$1.02 - 0.46 \cdot p^{0.67} + 3.28 \cdot 10^{-2} \cdot p^{0.67} \log n$	6.3%	$12{,}000 \times 12{,}000$
	E_{cf}	$1.05 - 0.48 \cdot p^{0.67} + 3.49 \cdot 10^{-2} \cdot p^{0.67} \log n$	6.1%	$11{,}000 \times 11{,}000$
	E_{ub}	$\min\{1, (5.8 \cdot 10^{-5} \cdot n^{1.75} \log n) \cdot p^{-1}\}$	1.7%	$7{,}800 \times 7{,}800$
Strassen	E_{ac}	$1.55 - 1.02 \cdot p^{0.25} + 4.59 \cdot 10^{-2} \cdot p^{0.25} \log n$	9.5%	$83{,}600 \times 83{,}600$
	E_{cf}	$1.26 - 0.65 \cdot p^{0.33} + 3.89 \cdot 10^{-2} \cdot p^{0.33} \log n$	5.9%	$12{,}680 \times 12{,}680$
	E_{ub}	$\min\{1, (0.25 \cdot n^{0.75}) \cdot p^{-1}\}$	2.4%	$1{,}200 \times 1{,}200$

By analyzing the discrepancies between these efficiency functions, we are able to provide answers to questions regarding co-design aspects, the connection between poor scaling and resource contention, optimization potential, and the presence of scalability bugs.

Discussion Our approach is viable for analyzing both the effects of resource contention on efficiency and further optimization potential. It provides users with an insight into whether the obstacle to scaling is resource contention or insufficient parallelism in the structure of the task dependency graph. In addition, users can also calculate the required input sizes to keep efficiency constant on a given core count. This approach can be used in co-design analysis to understand how many processing elements to put in a future machine, such that we can have high efficiency with realistic application input.

6 Ongoing Work

While Extra-P is already a powerful and versatile tool, we believe there are still some areas where we can improve and streamline our approach even more. Our efforts are currently focused on methods that would make it easier to consider multiple parameters in the performance modeling process. In the following, we briefly discuss two promising and so far unpublished approaches, one aiming to reduce

the number of inputs the modeling algorithm requires to generate accurate multi-parameter models, and another targeting the use of compile-time information to identify relevant parameters and formulate expectations regarding their interaction, further reducing the need for measurements while improving the quality of the resulting models.

6.1 Reducing the Cost of Measurements with Sparse Modeling

We have shown how useful the performance models we generate can be to developers, but even though the modeling is done cheaply and automatically, we still require a series of small-scale experiments in order to start the process. Therefore, the experiment design determines the quality of the model as well as the overall cost of the modeling process. The current state of the art requires at least five different values for each parameter, and measurements with all possible combinations of values for all parameters considered. Therefore, an exponential number of samples is needed, namely 5^n if n parameters are being modeled. For specific applications this makes it impractical to even create performance models. We are working on a novel parameter sampling approach that utilizes reinforcement learning, and leverages a sparse modeling technique, which only needs a polynomial number of samples and allows a more flexible experiment design.

We have made the observation that Extra-P assumes that there is one and only one behavior with respect to each parameter across the entire measured space. If this is true, the same function terms describing the effect of a given parameter should be identified no matter which sequence of five measurements is considered as long as the effect of all other parameters are kept constant. Rather than requiring all combinations of all values for each parameter, it could be sufficient to select a sequence of five measurements for each parameter to create single-parameter models, but a thorough analysis is required to ensure that lowering the number and cost of measurements is not detrimental to the quality of the results. When considering the interaction of parameters a new challenge arises: the binary decision of whether effect of any parameter pair is additive or multiplicative cannot be made with only a sequence of five measurements for each parameter. At least one additional data point is required, one that is not part of those sequences. In our evaluation, the addition of this one additional point improves the number of correctly identified models from 81.1 to 99.9%, while more data points only marginally improved the results.

Figure 4 shows a set of measurements that is usually sufficient to correctly identify two-parameter performance models. Of course, any of the columns and rows could be used to generate the performance model. The question as to how to select which rows and columns to measure as well as which additional points to consider such that the best models can be generated with the smallest cost is still open. While selecting the combination with the smallest cost is appealing, we must quantify how the quality of the models degrades compared to other strategies. For

www.dbooks.org

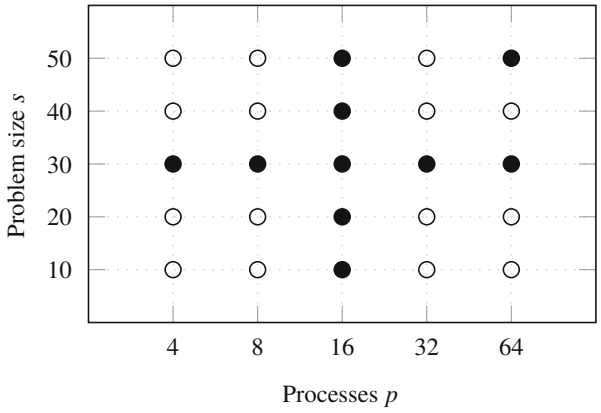


Fig. 4 Example set of performance measurements for a two-parameter analysis. The filled circles represent a subset that is likely to be sufficient to create a performance model

this purpose, we leverage reinforcement learning to compare and evaluate different strategies.

Using this approach on some of our existing case studies shows very promising preliminary results: we were able to reduce the average modeling costs by up to 93% while maintaining 99% of the model accuracy. We are currently in the process of analyzing the limits of this approach and trying to define if there are any conditions required for it to be successful, or if it is a valid solution for most applications.

6.2 Taint Analysis for Many-Parameter Modeling

The current workflow of Extra-P follows three major steps: parameter identification, designing a set of experiments to measure the influence of changes in the parameters of interest, and estimating the best model from the provided data. A lot of work in Catwalk and ExtraPeak focused on automating and improving the third step in this process, leaving the user to still identify parameters of interest and choose which values to give these parameters for the subsequent measurements that will serve as input to Extra-P.

We are prototyping a tool called perf-taint to alleviate this issue. Perf-taint is an LLVM-based hybrid program analysis integrated with Extra-P that will supply program information to the modeling process. We use taint analysis [14], a computer security technique which reliably relates marked input values with the program parts they potentially affect to determine which parameters influence the performance critical control-flow in the program and detects functions that are constant with respect to selected performance parameters.

We have preliminary results using two benchmarks: LULESH and su3_rmd from the MILC suite. In LULESH, we consider three parameters and prune 303 out of 347 functions. In su3_rmd, we consider four parameters and prune 364 out of

621 functions. The instrumentation overhead is decreased 45 times in the case of LULESH compared to a full instrumentation. While these massive improvements in runtime overhead are likely specific to object-oriented C++ applications, the quality of the resulting models is increased across all our experiments. We hope this approach will allow better models to be generated with even less effort even for large, complex applications where the importance of individual parameters is not necessarily well understood.

7 Related Work

Performance analysis and prediction of real-world application workloads is most important in high-performance computing. Performance tools such as Score-P [1] allow the programmer to observe the performance of real-world applications at impressive scales but are often limited to observations of the current configuration and do not provide insight into their behavior when being scaled further.

Such insights can be obtained with the help of analytical performance models, which have a long history. Early manual models showed to be very effective in describing application performance characteristics [27] and understanding complex behaviors [33]. Hoefler et al. established a simple six-step process to guide the (manual) creation of analytical performance models [21]. The resulting models lead to interesting insights into application behavior at scale and on unknown systems [2]. The six-step process formed the blueprint of our own approach.

Various automated performance modeling methods exist. Tools such as PALM [43] use extensive and detailed per-function measurements to build structural performance models of applications. The creation of structural models is also supported by dedicated languages such as Aspen [42]. These methods are powerful but require the prior manual annotation of the source code.

Hammer et al. combine static source-code analysis with cache-access simulations to create ECM and roofline models of steady-state loop kernels [19]. While their approach uses hardware information gathered on the target machine, it does not actually run the code but relies on static information instead. Lo et al. create roofline models for entire applications automatically and attempt to identify the optimal configuration to run an application on a given system [29]. Extra-P, in contrast, identifies scalability bugs in individual parts of an application rather than determining the optimal runtime configuration on a particular system.

Vuduc et al. propose a method of selecting the best implementation for a given algorithm by automatically generating a large number of candidates for a selected kernel and then choosing the one offering the best performance according to the results of an empirical search [45]. Our approach generates performance models for *all* kernels in a given application to channel the optimization efforts to where they will be most effective. Zaparanakus et al. analyze and group loops and repetitions in applications towards automatically creating performance profiles for sequential algorithms [49]. Goldsmith et al. use clustering and linear regression analysis to

www.dbooks.org

derive performance model coefficients from empirical measurements [16]. This approach requires the user to define either a linear or power law expectation for the performance model unlike the greater freedom offered by the performance model normal form defined in our approach. Jayakumar et al. predict runtimes of entire applications automatically using machine-learning approaches [25].

Zhai, Chen, and Zheng extrapolate single-node performance of applications with a known regular structure to complex parallel machines via simulation [50], but require the entire memory that would be needed at the target scale to correctly extrapolate performance. Wu and Müller [48] showed how to predict the communication behavior of stencil codes at larger scales by extrapolating their traces. While still requiring an SPMD-style parallel execution paradigm, Extra-P has proven to work with general OpenMP or MPI codes beyond pure stencil codes.

Carrington et al. introduced a model-based performance prediction framework for applications on different computers [13]. Marin and Mellor-Crummey utilize semi-automatically derived performance models to predict performance on different architectures [30]. Siegmund et al. analyze the interaction of different configuration options and model how this affects the performance of an application as a whole rather than looking at its individual components [18, 41].

Reducing the burden of collecting the measurements required for the empirical learning process is a research effort in its own right. Sarkar et al. [36] suggest a powerful sampling approach which can be used if all features of interest are boolean. Another approach for sampling highly configurable systems with boolean configuration options by Zhang et al. [51] suggests using the Fourier transform to select the best samples. However, these methods cannot be directly adapted to our use case: the features modeled by Extra-P are allowed a much wider range of expression. They can be not just boolean, but functions with polynomial and logarithmic terms.

8 Conclusion

In the Catwalk project, we initially set out to prove that automated performance modeling is feasible and that automatically generated models are accurate enough to identify scalability bugs. We started by showing that in those cases where hand-crafted models existed in the literature our models are competitive. Our interaction with many different users from different fields taught us that approximate models are acceptable as long as the effort to create them is low and they do not mislead. Furthermore, being able to produce many performance models cheaply helps drastically improve code coverage, which is as important as model accuracy. Having approximate models for all parts of the code can be more useful than having a model with 100% accuracy for just a tiny portion of the code or no model at all.

Finally, after the public release of the Extra-P software and numerous tutorials where Extra-P was introduced, we have seen growing interest from HPC application developers—whether for immediate use or in incorporating Extra-P in their own

research. The continuous development of Extra-P during the ExtraPeak project was partly driven by feature requests from the users themselves, and while the collaboration uncovered many challenges, the results invariably proved useful beyond the problem they were specifically developed to solve: The capability of modeling the impact of multiple parameters simultaneously paved the way for complex approaches such as using application-centric requirements in the co-design process or determining the iso-efficiency of task-based parallel applications. We confidently claim that Extra-P is a powerful tool capable of providing insightful performance information for most developers while requiring only a modicum of experience in performance analysis and few resources.

Acknowledgments This work has been funded by the German Research Foundation (DFG) and the Swiss National Science Foundation (SNF), primarily through the DFG Priority Programme 1648 Software for Exascale Computing (SPPEXA) and the ExtraPeak project (Grant Nr. WO 1589/8-1) but also through the DFG Program Performance Engineering for Scientific Software. It has received further support from the Federal Ministry of Education and Research BMBF under Grant No. 01IH16008, the Hessian LOEWE initiative within the Software-Factory 4.0 project, and the US Department of Energy under Grant No. DE-SC0015524. Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt, Vulcan at Lawrence Livermore National Laboratory, and SuperMUC at Leibniz Supercomputing Centre.

References

1. an Mey, D., Biersdorff, S., Bischof, C., Diethelm, K., Eschweiler, D., Gerndt, M., Knüpfer, A., Lorenz, D., Malony, A.D., Nagel, W.E., Oleynik, Y., Rössel, C., Saviankou, P., Schmidl, D., Shende, S.S., Wagner, M., Wesarg, B., Wolf, F.: Score-P: a unified performance measurement system for petascale applications. In: Proceedings of the CiHPC: Competence in High Performance Computing, HPC Status Konferenz der Gauß-Allianz e.V., Schwetzingen, Germany, June 2010, Gauß-Allianz, pp. 85–97. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-24025-6_8
2. Bauer, G., Gottlieb, S., Hoefler, T.: Performance modeling and comparative analysis of the MILC lattice QCD application su3_rmd. In: 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), pp. 652–659. IEEE, Piscataway (2012)
3. Berg, E.: Efficient and flexible characterization of data locality through native execution sampling. Ph.D. Thesis, Department of Information Technology, Uppsala University (2005)
4. Bhattacharyya, A., Hoefler, T.: PEMOGEN: automatic adaptive performance modeling during program runtime. In: Proceedings of the 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT'14). ACM, Edmonton (2014)
5. Bischof, C., An Mey, D., Iwainsky, C.: Brainware for green HPC. Comput. Sci. Res. Dev. **27**(4), 227–233 (2012)
6. Blumofe, R.D., Joerg, C.F., Kuszmaul, B.C., Leiserson, C.E., Randall, K.H., Zhou, Y.: Cilk: an efficient multithreaded runtime system. J. Parallel Distrib. Comput. **37**(8), 55–69 (1997)
7. Brebner, P.C.: Automatic performance modelling from application performance management (APM) data: an experience report. In: Proceedings of the 7th ACM/SPEC on International Conference on Performance Engineering, ICPE '16, pp. 55–61. ACM, New York (2016). https://doi.org/10.1145/2851553.2851560
8. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A portable programming interface for performance evaluation on modern processors. Int. J. High Perform. Comput. Appl. **14**(3), 189–204 (2000)

www.dbooks.org

9. Burger, M., Bischof, C., Calotoiu, A., Wunderer, T., Wolf, F.: Exploring the performance envelope of the LLL algorithm. In: Proceedings of the 2018 IEEE International Conference on Computational Science and Engineering (CSE), pp. 36–43 (2018). https://doi.org/10.1109/CSE.2018.00012
10. Calotoiu, A., Hoefler, T., Poke, M., Wolf, F.: Using automated performance modeling to find scalability bugs in complex codes. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC13), Denver, pp. 1–12. ACM, New York (2013). https://doi.org/10.1145/2503210.2503277
11. Calotoiu, A., Beckinsale, D., Earl, C.W., Hoefler, T., Karlin, I., Schulz, M., Wolf, F.: Fast multi-parameter performance modeling. In: Proceedings of the 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp. 172–181 (2016). https://doi.org/10.1109/CLUSTER.2016.57
12. Calotoiu, A., Graf, A., Hoefler, T., Lorenz, D., Rinke, S., Wolf, F.: Lightweight requirements engineering for exascale co-design. In: Proceedings of the 2018 IEEE International Conference on Cluster Computing (CLUSTER), pp. 201–211 (2018). https://doi.org/10.1109/CLUSTER.2018.00038
13. Carrington, L., Snavely, A., Wolter, N.: A performance prediction framework for scientific applications. Futur. Gener. Comput. Syst. **22**(3), 336–346 (2006). https://doi.org/10.1016/j.future.2004.11.019
14. Clause, J., Li, W., Orso, A.: Dytan: a generic dynamic taint analysis framework. In: Proceedings of the 2007 International Symposium on Software Testing and Analysis, ISSTA '07, pp. 196–206. ACM, New York (2007). https://doi.org/10.1145/1273463.1273490
15. Galakatos, A., Markovitch, M., Binnig, C., Fonseca, R., Kraska, T.: Fiting-tree: a data-aware index structure. In: Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, June 30–July 5, pp. 1189–1206 (2019). https://doi.org/10.1145/3299869.3319860
16. Goldsmith, S.F., Aiken, A.S., Wilkerson, D.S.: Measuring empirical computational complexity. In: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC-FSE '07, pp. 395–404. ACM, New York (2007). https://doi.org/10.1145/1287624.1287681
17. Grama, A.Y., Gupta, A., Kumar, V.: Isoefficiency: measuring the scalability of parallel algorithms and architectures. Parallel Distrib. Technol. Syst. Appl. **1**(3), 12–21 (1993)
18. Grebhahn, A., Rodrigo, C., Siegmund, N., Gaspar, F.J., Apel, S.: Performance-influence models of multigrid methods: a case study on triangular grids. Concurr. Comp. Pract. Exp. **29**(17), e4057 (2017)
19. Hammer, J., Hager, G., Eitzinger, J., Wellein, G.: Automatic loop kernel analysis and performance modeling with Kerncraft. CoRR abs/1509.03778 (2015). https://doi.org/10.1145/2832087.2832092
20. Hoefler, T., Janisch, R., Rehm, W.: Parallel scaling of Teter's minimization for Ab Initio calculations. In: HPC Nano'06 in Conjunction with the International Conference on High Performance Computing, Networking, Storage and Analysis, SC06 (2006)
21. Hoefler, T., Gropp, W., Kramer, W., Snir, M.: Performance modeling for systematic performance tuning. In: State of the Practice Reports, SC '11, pp. 6:1–6:12. ACM, New York (2011). https://doi.org/10.1145/2063348.2063356
22. Hoefler, T., Kwasniewski, G.: Automatic complexity analysis of explicitly parallel programs. In: Proceedings of the 26th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '14, pp. 226–235. ACM, New York (2014). https://doi.org/10.1145/2612669.2612685
23. Ilyas, M.K., Calotoiu, A., Wolf, F.: Off-road performance modeling – how to deal with segmented data. In: European Conference on Parallel Processing, vol. 10417, pp. 36–48. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-64203-1_3
24. Iwainsky, C., Shudler, S., Calotoiu, A., Strube, A., Knobloch, M., Bischof, C., Wolf, F.: How many threads will be too many? On the scalability of OpenMP implementations. In: Proceedings of the 21st Euro-Par Conference, Vienna, Austria. Lecture Notes in Computer Science, vol. 9233, pp. 451–463. Springer, Berlin (2015). https://doi.org/10.1007/978-3-662-48096-0_35

25. Jayakumar, A., Murali, P., Vadhiyar, S.: Matching application signatures for performance predictions using a single execution. In: 2015 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1161–1170 (2015). https://doi.org/10.1109/IPDPS.2015.20
26. Jülich Supercomputing Centre: JuBE - Jülich Benchmarking Environment (2016). http://www.fz-juelich.de/jsc/jube
27. Kerbyson, D.J., Alme, H.J., Hoisie, A., Petrini, F., Wasserman, H.J., Gittings, M.: Predictive performance and scalability modeling of a large-scale application. In: Proceedings of the ACM/IEEE Conference on Supercomputing (SC'01), p. 37. ACM, New York (2001). http://doi.acm.org/10.1145/582034.582071
28. Kunen, A.J.: Kripke - user manual v1.0. Technical Report, LLNL-SM-658558, Lawrence Livermore National Laboratory (2014)
29. Lo, Y.J., Williams, S., Van Straalen, B., Ligocki, T.J., Cordery, M.J., Wright, N.J., Hall, M.W., Oliker, L.: Roofline model toolkit: a practical tool for architectural and program analysis. In: High Performance Computing Systems. Performance Modeling, Benchmarking, and Simulation, pp. 129–148. Springer, Berlin (2014)
30. Marin, G., Mellor-Crummey, J.: Cross-architecture performance predictions for scientific applications using parameterized models. SIGMETRICS Perform. Eval. Rev. **32**(1), 2–13 (2004). https://doi.org/10.1145/1012888.1005691
31. Mathis, M.M., Amato, N.M., Adams, M.L.: A general performance model for parallel sweeps on orthogonal grids for particle transport calculations. Technical Report, College Station, TX (2000)
32. OpenMP Architecture Review Board: OpenMP application programming interface, version 4.0 (2013). http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
33. Pllana, S., Brandic, I., Benkner, S.: Performance modeling and prediction of parallel and distributed computing systems: a survey of the state of the art. In: Proceedings of the 1st International Conference on Complex, Intelligent and Software Intensive Systems (CISIS), pp. 279–284 (2007). https://doi.org/10.1109/CISIS.2007.49
34. Reisert, P., Calotoiu, A., Shudler, S., Wolf, F.: Following the blind seer – creating better performance models using less information. In: European Conference on Parallel Processing, vol. 10417, pp. 106–118. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-64203-1_8
35. Rinke, S., Butz-Ostendorf, M., Hermanns, M.A., Naveau, M., Wolf, F.: A scalable algorithm for simulating the structural plasticity of the brain. J. Parallel Distrib. Comput. **120**, 251–266 (2018). https://doi.org/10.1016/j.jpdc.2017.11.019
36. Sarkar, A., Guo, J., Siegmund, N., Apel, S., Czarnecki, K.: Cost-efficient sampling for performance prediction of configurable systems. In: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), ASE '15, pp. 342–352. IEEE Computer Society, Washington (2015). https://doi.org/10.1109/ASE.2015.45
37. Shudler, S., Calotoiu, A., Hoefler, T., Strube, A., Wolf, F.: Exascaling Your library: will your implementation meet your expectations? In: Proceedings of the 29th ACM on International Conference on Supercomputing, ICS '15, pp. 165–175. ACM, New York (2015). https://doi.org/10.1145/2751205.2751216
38. Shudler, S., Calotoiu, A., Hoefler, T., Wolf, F.: Isoefficiency in practice: configuring and understanding the performance of task-based applications. In: Proceedings of the 22nd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '17, pp. 131–143. ACM, New York (2017). https://doi.org/10.1145/3018743.3018770
39. Shudler, S., Berens, Y., Calotoiu, A., Hoefler, T., Strube, A., Wolf, F.: Engineering algorithms for scalability through continuous validation of performance expectations. IEEE Trans. Parallel Distrib. Syst. **30**(8), 1768–1785 (2019). https://doi.org/10.1109/TPDS.2019.2896993
40. Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G.: SPL conqueror: toward optimization of non-functional properties in software product lines. Softw. Qual. J. **20**(3), 487–517 (2012). https://doi.org/10.1007/s11219-011-9152-9

www.dbooks.org

41. Siegmund, N., Grebhahn, A., Apel, S., Kästner, C.: Performance-influence models for highly configurable systems. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 284–294. ACM, New York (2015). https://doi.org/10.1145/2786805.2786845
42. Spafford, K.L., Vetter, J.S.: Aspen: a domain specific language for performance modeling. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '12, pp. 84:1–84:11. IEEE Computer Society Press, Los Alamitos (2012). http://dl.acm.org/citation.cfm?id=2388996.2389110
43. Tallent, N.R., Hoisie, A.: Palm: easing the burden of analytical performance modeling. In: Proceedings of the 28th ACM International Conference on Supercomputing, ICS '14, pp. 221–230. ACM, New York (2014). https://doi.org/10.1145/2597652.2597683
44. Vogel, A., Calotoiu, A., Nägel, A., Reiter, S., Strube, A., Wittum, G., Wolf, F.: Automated performance modeling of the ug4 simulation framework. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing – SPPEXA 2013-2015, pp. 467–481. Springer International Publishing, Cham (2016)
45. Vuduc, R., Demmel, J.W., Bilmes, J.A.: Statistical models for empirical search-based performance tuning. Int. J. High Perform. Comput. Appl. **18**(1), 65–94 (2004). https://doi.org/10.1177/1094342004041293
46. Wang, W., Dey, T., Davidson, J.W., Soffa, M.L.: DraMon: predicting memory bandwidth usage of multi-threaded programs with high accuracy and low overhead. In: Proceedings of the 2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA '14), pp. 380–391 (2014)
47. Wolf, F., Bischof, C., Calotoiu, A., Hoefler, T., Iwainsky, C., Kwasniewski, G., Mohr, B., Shudler, S., Strube, A., Vogel, A., Wittum, G.: Automatic performance modeling of HPC applications. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing - SPPEXA 2013-2015, pp. 445–465. Springer International Publishing, Cham (2016)
48. Wu, X., Müller, F.: Scalaextrap: trace-based communication extrapolation for SPMD programs. ACM Trans. Program. Lang. Syst. **34**(1) (2012)
49. Zaparanuks, D., Hauswirth, M.: Algorithmic profiling. In: Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, pp. 67–76. ACM, New York (2012). https://doi.org/10.1145/2254064.2254074
50. Zhai, J., Chen, W., Zheng, W.: Phantom: predicting performance of parallel applications on large-scale parallel machines using a single node. SIGPLAN Not. **45**(5), 305–314 (2010). https://doi.org/10.1145/1837853.1693493
51. Zhang, Y., Guo, J., Blais, E., Czarnecki, K.: Performance prediction of configurable software systems by Fourier learning. In: Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), ASE '15, pp. 365–373. IEEE Computer Society, Washington (2015). https://doi.org/10.1109/ASE.2015.15

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



FFMK: A Fast and Fault-Tolerant Microkernel-Based System for Exascale Computing



Carsten Weinhold, Adam Lackorzynski, Jan Bierbaum, Martin Küttler, Maksym Planeta, Hannes Weisbach, Matthias Hille, Hermann Härtig, Alexander Margolin, Dror Sharf, Ely Levy, Pavel Gak, Amnon Barak, Masoud Gholami, Florian Schintke, Thorsten Schütt, Alexander Reinefeld, Matthias Lieber, and Wolfgang E. Nagel

Abstract The FFMK project designs, builds and evaluates a system-software architecture to address the challenges expected in Exascale systems. In particular, these challenges include performance losses caused by the much larger impact of runtime variability within applications, hardware, and operating system (OS), as well as increased vulnerability to failures. The FFMK OS platform is built upon a multi-kernel architecture, which combines the L4Re microkernel and a virtualized Linux kernel into a noise-free, yet feature-rich execution environment. It further includes global, distributed platform management and system-level optimization services that transparently minimize checkpoint/restart overhead for applications. The project also researched algorithms to make collective operations fault tolerant in presence of failing nodes. In this paper, we describe the basic components, algorithms, and services we developed in Phase 2 of the project.

1 Introduction

The operating system (OS) abstracts from low-level aspects of a computer system's hardware by providing applications with standardized programming interfaces and common services such as file systems and network access. By design, it

C. Weinhold (✉) · A. Lackorzynski · J. Bierbaum · M. Küttler · M. Planeta · H. Weisbach · M. Hille · H. Härtig · M. Lieber · W. E. Nagel
TU Dresden, Dresden, Germany
e-mail: carsten.weinhold@tu-dresden.de

A. Margolin · D. Sharf · E. Levy · P. Gak · A. Barak
The Hebrew University of Jerusalem, Jerusalem, Israel

M. Gholami · F. Schintke · T. Schütt · A. Reinefeld
Zuse Institute Berlin, Berlin, Germany

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_16

www.dbooks.org

stands between the hardware and all applications. In the high-performance computing (HPC) community, the OS is therefore sometimes considered to be "in the way" as applications try to extract maximum performance from the underlying hardware. Indeed, the OS can introduce overhead, as we will discuss in the following. But challenges posed by upcoming Exascale systems such as load imbalances or failures due to increasing component counts can benefit from system-level support. Therefore, the central goal of the FFMK project has been to investigate how the OS can actually help, rather than be a source of overhead.

In the following paragraphs, we summarize the general architecture of the FFMK OS platform and give an overview of its higher-level services. In part, this description is an overview of results from Phase 1 of the project; but it shall also help put the results presented in this paper into context. For a much more detailed discussion of FFMK and the motivation behind it, we refer to our previous project report [59].

Multi-Kernel Node OS Figure 1 shows the architecture of the FFMK node OS. It is built on a multi-kernel foundation comprising an L4 microkernel and a variant of the Linux kernel that is called L^{4}Linux. We aim to support unmodified HPC applications and they shall have access to the same runtime libraries and communication drivers that are used on standard Linux-based HPC OSes. We target noise-sensitive applications by providing jitter-free execution directly on top our microkernel [33]. In this context, we also investigated the influence of hardware performance variation [61]. However, our vision for an HPC OS platform also includes new platform management services to support more complex and

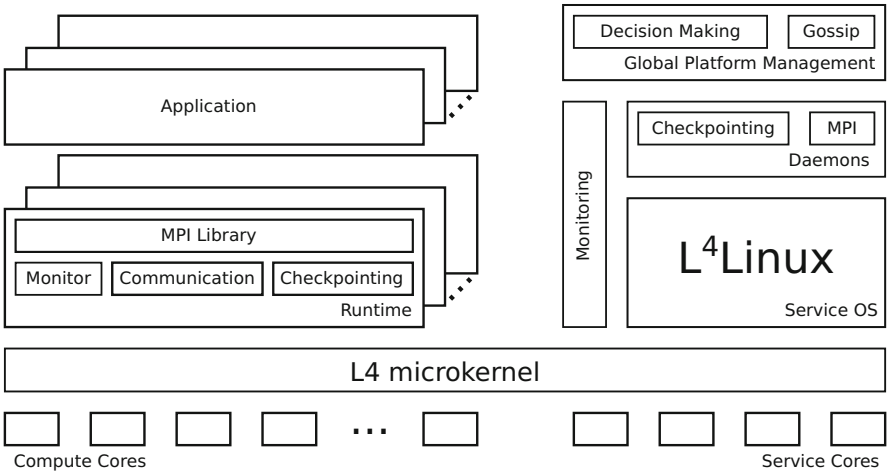


Fig. 1 FFMK software architecture: compute processes with performance-critical parts of (MPI) runtime and communication driver execute directly on L4 microkernel; functionality that is not critical for performance is offloaded to the L^{4}Linux kernel, which also hosts global platform management and fault-tolerance services

dynamic applications, as well as algorithms and system-level support to address fault-tolerance challenges posed by Exascale systems with unprecedented hardware component counts.

Applications, Runtimes, Communication HPC applications are highly specialized, but they achieve a certain level of platform independence by using common runtime and communication APIs such as the Message Passing Interface (MPI) [18]. However, just proving an MPI library and interconnect drivers (e.g., for InfiniBand) is not sufficient [60], because the majority of HPC codes use many Linux-specific APIs, too. The same is true for most HPC infrastructure, including parallel file systems and cluster management solutions. Compatibility to Linux is therefore essential and this can only be achieved, if applications are in fact compiled for Linux and started as Linux processes.

Dynamic Platform Management The FFMK OS platform is more than a multi-kernel architecture. As motivated in the Phase 1 report [59], we include distributed global management, because the system software is best suited to monitor health and load of nodes. This is in contrast to the current way of operating HPC clusters and supercomputers, where load balancing problems and fault tolerance are tasks that practically *every* application deals with on its own. In the presence of frequent component failures, hardware heterogeneity, and dynamic resource demands, applications can no longer assume that compute resources are assigned statically.

Load Balancing We aim to shift more coordination and decision making into the system layer. In the FFMK OS, the necessary monitoring and decision making is done at three levels: (1) on each node, (2) per application instance across multiple nodes, and (3) based on a global view by redundant master management nodes. We published fault-tolerant gossip algorithms [3] suitable for inter-node information dissemination and found that they have negligible performance overhead [36]. We further achieved promising results with regard to oversubscribing of cores, which can improve throughput for some applications [62]. We have since integrated the gossip algorithm, a per-node monitoring daemon, and a distributed decision making algorithm aimed at automatic, process-level load balancing for oversubscribed nodes. However, one key component of this platform management service is still missing: the ability to migrate processes from overloaded nodes to ones that have spare CPU cycles. Transparent migration of MPI processes that directly access InfiniBand hardware has proven to be extremely difficult. We leave this aspect for a future publication, but do we do summarize our key results on novel diffusion-based load balancing algorithms [39] in this report. These algorithms could be integrated into the FFMK load management service once process-level migration is possible.

Fault Tolerance The ability to migrate processes away from failing nodes can also be used for proactive fault tolerance. However, the focus of our research on system-level fault tolerance has been in two other areas. First, we published on efficient collective operations in the presence of failures [27, 31, 43]. Second, we continued research on scalable checkpointing, where we concentrated on global coordination

www.dbooks.org

for user-level checkpointing [22] and how to optimize it based on expected failure rates [21].

In the following two sections of this paper, we discuss how re-architecting the OS kernel for HPC systems can improve performance and scalability (Sect. 2); we also present results on how to increase kernel scalability beyond the dozens of cores found in contemporary systems, as well as new load balancing algorithms. We then make the case that system-software support is essential to address fault-tolerance challenges posed by Exascale systems and how new fault-tolerant algorithms can help improve robustness and performance of collective operations (Sect. 3). Each individual subsection on these pages summarizes our peer-reviewed work in the respective area.

2 Building Blocks for a Scalable HPC Operating System

2.1 The Case for a Multi-Kernel Operating System

Noise-Sensitive Applications A widely reported problem in HPC is "OS noise" [5, 16, 26, 49, 53], where sporadic or periodic housekeeping activities of the OS (or other background tasks) briefly interrupt application threads. These interruptions can slow down applications based on the bulk-synchronous programming (BSP) model. BSP applications are parallel programs that are characterized by alternating computation and communication phases that all participating threads must perform in perfect synchronization to maximize throughput. If just a few compute threads are preempted by background activities, all other threads that depend on their input will waste CPU cycles as they wait for the stragglers to reach the communication phase. As shown in Fig. 2, delays can amount to hundreds of thousands of cycles, resulting in a slowdown of up to 9% for a computation that takes 1.5 ms to complete when there is no interruption.

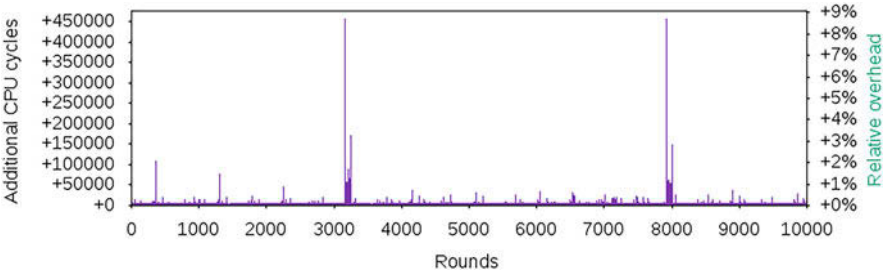


Fig. 2 OS noise during a run of the fixed work quantum (FWQ) benchmark on a node of a production HPC cluster with Linux-based vendor OS

Re-Architecting the Kernel for HPC One way to address the OS noise problem is to partition the compute resources of each node into two sets of cores: *compute cores* that are allocated exclusively to HPC applications and *service cores* that run management and maintenance tasks. Multi-kernel OS architectures implement this approach by assigning the compute cores to a lightweight kernel (LWK); a traditional kernel such as Linux runs management and node monitoring daemons on the service cores. The LWK does not preempt compute threads, thereby minimizing execution-time jitter for applications. However, a LWK for HPC does not replace a complex kernel such as Linux. It only implements functionality that is critical for application performance; system calls that do not impact performance are offloaded to the Linux kernel running on the service cores. The multi-kernel approach gives HPC applications the best of both worlds: the LWK ensures low noise and high performance, whereas Linux offers convenience, familiar APIs, a rich feature set, and compatibility with huge amounts of legacy infrastructure.

A Microkernel as a Universal LWK Multi-kernel OS architectures have received a lot of attention in recent years and several new LWKs have been developed, including IHK/McKernel [55] and mOS [63]. However, the maintenance effort for keeping an HPC-suitable multi-kernel OS up to date and compatible with Linux must be smaller than constantly patching mainline Linux to make it less noisy. Arguably, the best way to meet this requirement is to reuse components that already exist and that are actively maintained. In the FFMK project, we therefore chose to use the mature L4Re microkernel and L^{4}Linux as the basis of the FFMK node OS. In contrast to McKernel and mOS, the L4Re microkernel manages not just the designated compute cores, but all processor resources. The L^{4}Linux kernel runs virtualized on top of L4Re as shown in Fig. 1 on page 484.

2.2 *L4Re Microkernel and L^{4}Linux*

In this Subsection, we quote[1] from a previous publication [60] an overview of the L4Re ecosystem, including the L4Re microkernel and the paravirtualized Linux kernel L^{4}Linux. These two basic building blocks are combined into a foundation of a highly flexible and low-noise node OS. In Sect. 2.3, we evaluate the main benefits of this multi-kernel architecture.

L4 Microkernel The L4Re microkernel is a member of the L4 family of microkernels. The core principle of L4 [40] is that the kernel should provide only the minimal amount of functionality that is necessary to build a complete OS on top of it. Thus, an L4 microkernel is not intended to be a minimized Unix, but instead it provides only a few basic abstractions: address spaces, threads, and inter-

[1]This description has been shortened and slightly edited for brevity; see [60] for the complete version.

www.dbooks.org

process communication (IPC). For performance reasons, a thread scheduler is also implemented within the kernel. However, other OS functionality such as device drivers, memory management, or file systems are provided by system services running as user-level programs on top of the microkernel.

Applications and User-Level Services L4Re applications communicate with each other and with system services by exchanging IPC messages. These IPC messages can not only carry ordinary data, but they may also transfer access rights for resources. Being able to map memory pages via IPC allows any two programs to establish shared memory between their address spaces. Furthermore, because it is possible to revoke memory mappings at any time, this features enables user-level services to implement arbitrary memory-management policies. In much the same way an L4Re program can pass a *capability* referencing a resource to another application or service, thereby granting the receiver the permission to access that resource. A capability can refer to a kernel object such as a `Thread` or a `Task`, representing an independent flow of execution or an address space, respectively. But they may also point to an `Ipc_gate`, which is a communication endpoint through which any user-space program can offer an arbitrary service to whomever possesses the corresponding capability.

I/O Device Support An important feature of the L4Re microkernel is that it maps hardware interrupts to IPC messages. A thread running in user space can receive interrupts by waiting for messages from an `Irq` kernel object. In conjunction with the possibility to map I/O memory regions of hardware devices directly into user address spaces, it is possible to implement device drivers outside the microkernel.

Virtualized Linux The L4Re microkernel is a fully functional hypervisor capable of hosting virtual machines running unmodified guest operating systems. It employs hardware-assisted virtualization on instruction set architectures that support it, including x86, ARM, and MIPS. Device emulation or passthrough is supported through virtual machine monitors running in user space. However, faithful virtualization is not the only way to run a legacy OS on top of the L4Re microkernel. L^4Linux is a paravirtualized Linux kernel that has been adapted to run on the interfaces provided by L4Re. It is binary compatible with standard Linux programs, however, instead of running in the privileged mode of the CPU, the L^4Linux kernel runs as a multi-threaded user-level program. Linux user processes run in their own L4 tasks (i.e., other address spaces). Linux programs on L^4Linux experience the same protection as on native Linux; they cannot read or write the Linux kernel's memory and they are protected from each other like processes on native Linux.

The vCPU Mechanism For execution, L^4Linux employs vCPUs, a mechanism provided by the microkernel that allows for an asynchronous execution model where a vCPU migrates between executing code in the L^4Linux kernel and user code in Linux processes. For any event that needs to be handled by the Linux kernel, such as system calls and page faults by processes, or external interrupts by devices, the vCPU switches to the L^4Linux kernel to handle them.

L⁴Linux Process Model L⁴Linux manages the address spaces of Linux user processes through `Task` objects provided by the L4Re microkernel. Thus, every Linux process and the contents of its address space are known to the microkernel. Furthermore, L⁴Linux multiplexes all user-level threads executing in such an address space onto its vCPUs. Thus, the L4Re microkernel is involved in every context switch of any Linux user thread. In particular, it is responsible for forwarding any exceptions raised by a Linux program to the L⁴Linux kernel. Exceptions occur when a thread makes a system call, when a page fault occurs during its execution, or when a hardware device signals an interrupt. L⁴Linux receives these exceptions at a previously registered vCPU entry point, to which the microkernel switches the control flow when it migrates the vCPU from the `Task` of the faulting Linux user program to the address space of the virtualized Linux kernel.

2.3 Decoupled Execution on L⁴Linux

Since virtually all HPC codes are developed for Linux and require so many of its APIs, the only practical option is to execute them on a Linux-based OS. Just running them on L⁴Linux yields no benefit. However, the tight interaction between the L4Re microkernel and L⁴Linux allows us to conveniently implement a new mechanism we call *decoupling* [33].

Decoupling Thread Execution from L⁴Linux The purpose of decoupling is to separate execution of a thread in a Linux process from the vCPU it is normally running on. To this end, we create a separate, native L4Re host thread that runs in the same L4 `Task` (i.e., address space) as the Linux process, but not under control of L⁴Linux. The original Linux thread context in the L⁴Linux kernel is suspended while execution progresses on the native L4Re host thread. The user code running there will raise exceptions just as if it were executed by a vCPU, except that the microkernel forwards each of them to L⁴Linux as an *exception IPC message*. A message of this type carries a thread's register state and fault information as its payload, and is delivered by the microkernel to an exception handler. We configure L⁴Linux to be the exception handler of the "decoupled" Linux user threads. An attempt to perform a Linux system call will also result in an exception, which the L⁴Linux kernel can then handle by briefly reactivating the previously suspended thread context and scheduling it onto a vCPU. Afterwards, execution is resumed in decoupled mode on the L4Re host thread. Figure 3 visualizes decoupled execution; more details can be found our publications [33, 60].

One Mechanism for the Best of Both Worlds The net gain of the decoupling mechanism is that we can combine noise-free execution on our LWK (i.e., the L4Re microkernel) with the rich execution environment of Linux, including all its APIs and the HPC infrastructure built for it. Decoupled threads use a *single* mechanism for forwarding any system call or exception, instead of many specialized

www.dbooks.org

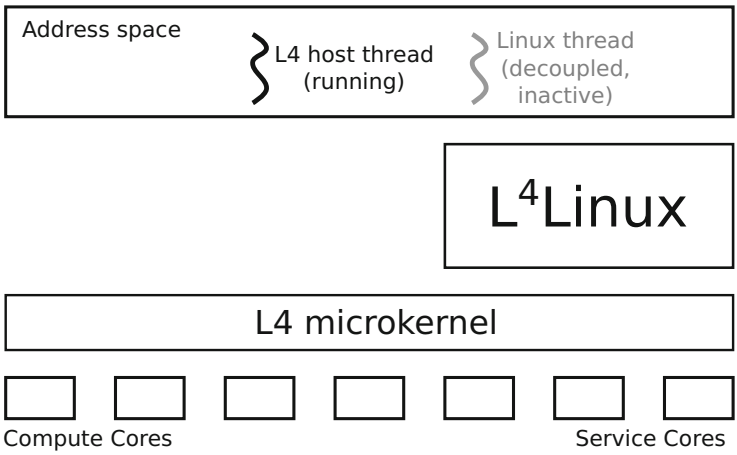


Fig. 3 Schematic view of the decoupling mechanism. The L4Re microkernel runs on every core of the system, while the virtualized L4Linux runs on a subset of those cores only. All normal Linux applications are thus restricted to those cores. Decoupling pulls off threads and runs them on cores not available to L4Linux

proxies that other multi-kernel HPC OSes use and that are difficult to maintain [60]. Applications are built for Linux and start running as Linux processes, but we pull their threads out of Linux's scheduling regime so they can run on dedicated cores without being disturbed by L4Linux. Effectively, decoupled threads run directly on the microkernel. However, they can use all services provided by L4Linux, which will continue to handle Linux system calls and resolve page faults. Also, since the InfiniBand driver in the L4Linux kernel maps the I/O registers of the HCA into the address space of each MPI rank, the high performance and minimal latency of the user-space part of the driver is not impaired; a decoupled thread can program performance-critical operations just like it would on native Linux.

CPU Resource Control The number of vCPUs assigned to L4Linux and their pinning to physical CPU cores determines how much hardware parallelism an L4Linux-based virtual machine can use. All other cores are exclusively under control of the L4Re microkernel and can therefore be allocated exclusively to decoupled threads of HPC application processes.

Initial Benchmark We used the fixed-work quantum (FWQ) [35] benchmark to determine how much "OS noise" decoupled threads experience. FWQ executes a fixed amount of work in a loop, which on a perfectly noise-free system should require a constant amount of CPU cycles to complete. Figure 4 shows the result of our first benchmark run, performed on a 2-socket machine from our lab. Using the `rdtsc` instruction, we measured delays of up to 55 CPU cycles per iteration when FWQ is executed by a decoupled thread on a dedicated core. The execution-time jitter is reduced to 4 cycles per iteration, when FWQ is offloaded to the second socket, while L4Linux is pinned to a single core of socket 1.

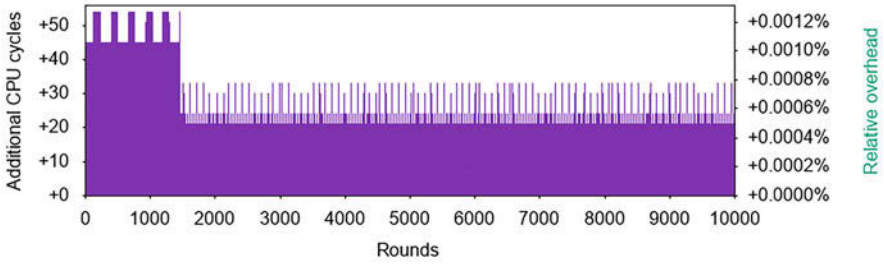


Fig. 4 Minimal OS noise remaining in decoupled execution; L^{4}Linux running on same socket

Multi-Node Benchmark The execution-time jitter we measured in the lab is four to fives orders of magnitude smaller than what we saw when FWQ ran on the vendor-provided Linux OS of Taurus, an HPC cluster installed at TU Dresden (see Fig. 2 on page 486). For larger-scale benchmarks, we extended FWQ into *MPI-FWQ*, a parallel benchmark that executes work-loop iterations in each MPI process on all participating cores. We performed experiments with our L4Re/L^{4}Linux-based node OS on 50 Taurus nodes. Each node has two Xeon® E5-2690 processors with 8 cores per socket. To benchmark "decoupling" in a parallel application, we allocated one core to L^{4}Linux and the remaining 15 cores to MPI-FWQ. The baseline we compare against is 15 MPI-FWQ processes per node scheduled by the same L^{4}Linux on the same hardware, but in a 16-vCPU configuration with no decoupled threads.

EP and BSP Runs MPI-FWQ can operate in two modes: In *StartSync* mode, a single barrier across all ranks is used to synchronize once when the benchmark starts; this mode simulates an embarrassingly parallel (EP) application. In *StepSync* mode, MPI-FWQ waits on a barrier after each iteration of the work loop, thereby acting like an application based on the bulk-synchronous programming (BSP) model.

Figure 5 shows the time to completion for any MPI-FWQ process operating in BSP-style StepSync mode, as we increased the total number of MPI processes (i.e., cores) from 30 to 750. The benchmark run time with decoupled MPI-FWQ threads (L4Linux-DC) is approximately 1% shorter than when the standard scheduler in the L^{4}Linux kernel controlled application threads (L4Linux-Std); results for EP-style StartSync runs show a similar performance. This difference is smaller than the up to 9% jitter we saw with the vendor OS, but our stripped down Linux environment lacks most of the management services and system daemons that run on the cluster. These services could never preempt *decoupled* threads, though.

More information on the decoupling mechanism, additional use-cases, and evaluations results can be found in separate publications [32–34].

www.dbooks.org

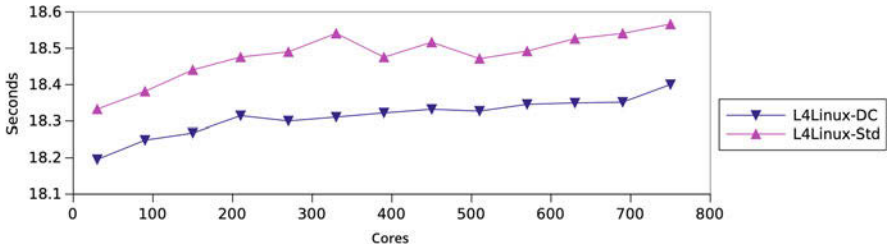


Fig. 5 BSP-style MPI-FWQ (*StepSync* mode) on L⁴Linux (Std) and with decoupled thread execution (DC) on Taurus. This figure has originally been published in [33]

2.4 Hardware Performance Variation

Software-induced imbalance in large-scale parallel simulations have also been studied by other groups, who proposed multi-kernel architectures as well [20, 23, 30, 48, 50, 51]. However, one source of variability has not been systematically investigated so far: the hardware itself. Hardware performance variation is particularly interesting due to growing diversity of platforms in HPC and the increasing complexity of computer architectures in general.

Measuring Hardware Performance Variation Characterizing hardware performance variability is challenging, because it requires a tightly controlled software environment. LWKs [51] have the greatest potential to obtain a precise characterization of various aspects of hardware performance variability on real HPC hardware. Towards this end, we have developed an extensible benchmarking framework to systematically characterize different aspects of hardware performance variability [61]. We use this benchmark suite to analyze five platforms described in Table 1: Intel Xeon [29], Intel Xeon Phi [56], Cavium ThunderX [9], Fujitsu FX100 [64] and IBM BlueGene/Q [28]. To minimize "OS noise", we benchmarked on OSes based on two LWKs: CNK on the IBM BG/Q and IHK/McKernel [20, 55] on the Intel, Fujitsu, and Cavium machines.

Benchmark Suite In addition to the previously described FWQ benchmark, our benchmark suite consists of seven other benchmark kernels. They were selected from well-known algorithms, micro benchmarks, or proxy applications and have the following characteristics:

- The **DGEMM** benchmark performs matrix multiplication. We confine ourselves to naïve matrix multiplication algorithms to allow compilers to emit SIMD instructions, if possible. This benchmark kernel is intended to measure hardware performance variation for double-precision floating point and vector operations. The **SHA** algorithm utilizes integer execution units instead.

Table 1 Summary of architectures

Platform/property	Intel Ivy Bridge	Intel KNL	Fujitsu FX100	Cavium ThunderX	IBM BG/Q
ISA	×86	×86	SPARC	ARM	PowerISA
Number of cores	8	64 + 4	32 + 2	48	16 + 2
Number of SMT threads	2	4	N/A	N/A	4
Clock frequency	2.6 GHz	1.4 GHz	2.2 GHz	2.0 GHz	1.6 GHz
L1d size	32 kB	32 kB	64 kB	32 kB	16 kB
L1i size	32 kB	32 kB	64 kB	78 kB	16 kB
L2 size	256 kB	1 MB × 34	24 MB	16 MB	32 MB
L3 size	20480 kB	N/A	N/A	N/A	N/A
On-chip network	Ring	2D mesh	unkown	Ring	Cross-bar
Process technology	22 nm	14 nm	20 nm	28 nm	45 nm

www.dbooks.org

- Using John McCalpin's **STREAM** benchmark, we assess variability in the cache and memory subsystems. The **Capacity** benchmark is intended to measure performance variation of cache misses themselves.
- **HACCmk** from the CORAL benchmark suite is a compute-intensive N-body simulation kernel with regular memory accesses. **HPCCG** from Mantevo's benchmark suite is a Mini-App aimed at exhibiting the performance properties of real-world physics codes working on unstructured grid problems. **MiniFE** is another proxy application for unstructured implicit finite-element codes from Mantevo's package.

More detailed descriptions of these benchmark kernels, modifications we made, and all measurement results can be found in our paper [61]. In this report, we highlight only a few key results from FWQ, HPCCG, and DGEMM experiments.

Workload Matters Like previous studies on software-induced performance variation, we relied on the FWQ benchmark to evaluate execution-time jitter for decoupled threads. However, this simple benchmark kernel, may be suitable to quantify interruptions caused by system software, but they are insufficient to capture hardware-induced noise. We hypothesize that the full extent of hardware performance variation can only be observed when the resources which cause these variations are actually used. And indeed, as shown in Fig. 6, the HPCCG proxy code running on IHK/McKernel on an Intel Ivy Bridge system shows about 1% of performance variation among all cores of a node.

We measured variation on 30 SMT cores of this 2-socket Intel Ivy Bridge E5-2650 v2 system for both FWQ and HPCCG. We set the working set size of HPCCG to 70% of the L1 data cache size (32 KiB), disabled TurboBoost, set the scaling governor to *performance*, and fixed the clock speed to the nominal frequency of 2.6 GHz. We additionally sampled the performance counters for L1 data cache and L1 instruction cache misses and confirmed that both benchmarks experience little to no misses, in particular cores one to seven and 16 to 29 experience neither instruction cache nor data cache misses under HPCCG. Nevertheless all cores show significantly more variation under HPCCG than under FWQ. We conclude that FWQ is indeed ill-suited to measure hardware-induced performance variation.

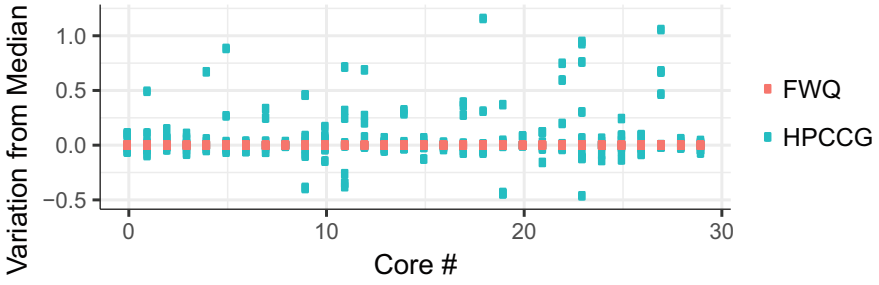


Fig. 6 Performance variation of FWQ and HPCCG on a dual-socket Intel E5-2650 v2

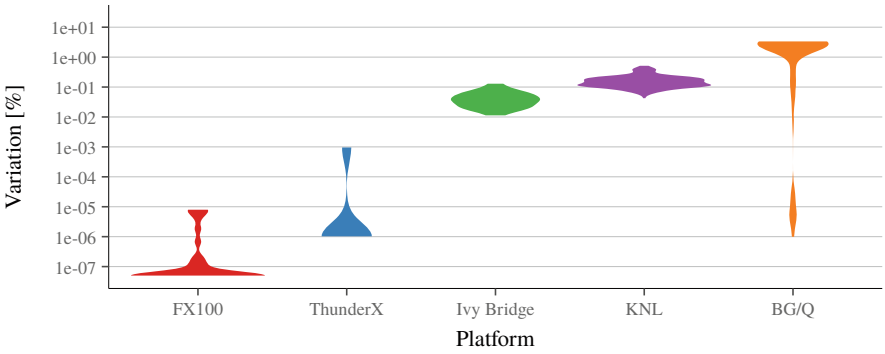


Fig. 7 Hardware performance variation under the DGEMM benchmark

Microarchitectures Differ Figure 7 visualizes our measurements for the DGEMM benchmark, which is dominated by floating point operations. We observe that the FX and ThunderX platforms exhibit very low variation, and note the rather high variation of the Ivy Bridge, KNL and BlueGene/Q platforms. We saw high numbers of cache misses on the Ivy Bridge platforms and therefore reduced the cache pressure to 70% fill level. After this modification to the benchmark setup, we saw stable or even zero cache miss numbers for all cores of the Ivy Bridge platform, but variation did not improve. We conclude that the measured variation is not caused by the memory subsystem.

Overall we found that just focusing on CPU core-local resources, like we did in this study, already shows up to six orders of magnitude difference in relative variation among different CPU architectures.

2.5 Scalable Diffusion-Based Load Balancing

A low-noise execution environment is essential for certain types of applications, as we explained in the preceding subsections. However, there are other HPC codes where noise is less of a concern, because they suffer from load imbalances that are inherent to the problem domain. For example, they might use multiple simulation kernels with different computational complexity. Furthermore, some computations are difficult to parallelize efficiently, because partitioning of the problem space is non-trivial; it might even change dynamically as the simulation progresses. As mentioned as part of the architecture overview in Sect. 1, we believe that support for load balancing should be provided at the system level, thereby freeing application developers from the burden of managing a dynamic system.

Taskifying MPI Applications may need to be (re-)balanced due to inherent load imbalances or to support shrinking and expanding the set of nodes assigned to

www.dbooks.org

the application. In the Phase 1 report [59] of this project, we proposed to *taskify* MPI processes using oversubscription, which results in multiple *Tasks* (i.e., MPI processes) per core that can be migrated transparently without application code modifications. MosiX-like algorithms [2] can be used for this kind of system-level load balancing in case communication between application tasks is insignificant. In other cases, repartitioning methods that consider the task communication graph are required [57].

Requirements for Efficient Load Balancing In the context of our system architecture, the requirements for a load-balancing method are: (1) effective load balancing with low inter-node communication (edge-cut), (2) low amount of migration to reduce MPI process migration costs, and (3) low overhead of the method itself. The method's input should be the local node's view of the weighted task communication graph, which can be obtained by monitoring the application at the OS/MPI level. Since graph partitioners, like ParMetis [52], are known to be computationally expensive, we developed a nearest-neighbor diffusion-based repartitioning method.

Method Description The method consists of four main phases that require point-to-point communication between neighbor nodes only and, with the exception of the flow calculation, have *O(number of tasks per node)* computational complexity.

1. Each node **determines** its **neighbor nodes** from the task communication graph.
2. **Flow calculation:** Computation of minimal load flows between neighbor nodes that lead to global balance using 2nd-order diffusion [13, 45]. The diffusion is stopped between each node pair individually if the load flow of an iteration falls below a specified threshold. The required number of iterations grows with the number of nodes. However, with low-latency networks the observed run time is within the low millisecond range even with thousands of nodes [39].
3. **Task selection:** Tasks at the partition border are selected for migration to realize workload flows using different weighted criteria to achieve low edge-cut [14].
4. **Partition refinement:** Edge-cut is improved with a parallel KL/FM-based refinement algorithm [58] that smoothes partition borders by swapping weighted tasks between neighbor node pairs independently within a certain imbalance tolerance. If the task selection result was not within the tolerance, the optimization goal is "balance" instead of "edge-cut".

Evaluation Workloads We implemented the diffusion method within the Zoltan load balancing library [12] and evaluate the performance on Taurus in a normal, non-oversubscribed setup. MPI processes own multiple migratable user-level tasks. Two scenarios with 3D task meshes are used for performance evaluation: the *Cloud scenario* consists of computation time measurements from $36 \times 36 \times 48$ grid cells over 100 time steps of COSMO-SPECS+FD4 [38] and the synthetic *Shock scenario* simulates the evolution of a spherical shock wave over a $160 \times 80 \times 80$ grid with a four times increased workload at the wave front over 169 time steps. Figure 8 shows the workload distribution for a 2D version of the shock scenario with two examples of resulting partitionings.



Fig. 8 Left: workload distribution for a selected time step of the 2D shock scenario, red indicates 4 times increased workload; Center: Resulting partitioning for 96 processes with ParMetis, colors represent individual partitions; Right: Resulting partitioning with Diffusion

Diffusion Results We compare our method with four other (re)partitioning methods: a hierarchical space-filling-curve method (FD4/SFC) [37], recursive coordinate bisection and space-filling curve from Zoltan (Zoltan/RCB and SFC), as well as AdaptiveRepart from ParMetis [52] with 0.5% imbalance tolerance. Note, that SFC and RCB are coordinate-based and not graph-based; they require application knowledge about spatial coordinates of tasks. Since diffusive load balancing requires a partitioning to exist, we use Zoltan/RCB to initialize the partitioning. Figure 9 shows the results for different metrics (lower is better), each averaged over the time steps (except median for run time):

- *Load Imbalance (max. load among processes/average load—1):* Diffusion performs better than ParMetis, but worse than the three geometric methods.
- *Migration (max. no. of tasks a process imports and exports/avg. no. of tasks per process):* Diffusion outperforms the other methods, especially with the Shock scenario (factor 3–10 less migration).

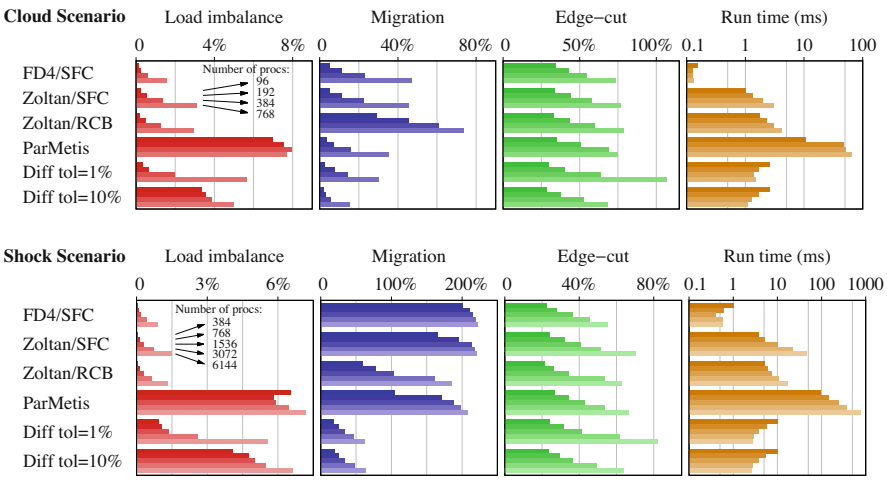


Fig. 9 Evaluation of diffusion-based load balancing with Cloud (62208 tasks) and Shock (ca. 1M tasks) scenarios. Bar shades show results for 96–768 and 384–6144 MPI processes, respectively

www.dbooks.org

- *Edge-cut (max. edge-cut among all processes/avg. number of edges per process):* All methods achieve very similar results, except diffusion with low imbalance tolerance at high process counts (coarse granularity).
- *Run time of the method (max. among all processes):* Due to its scalability, diffusion is clearly faster than the Zoltan methods at higher parallelism and 1–2 orders of magnitude faster than ParMetis. At 6144 processes, it requires 2.5 ms only.

As can be seen, the imbalance tolerance of the diffusion method allows to trade-off load balance vs. edge-cut and migration. We can conclude that our load balancing method enables fast, scalable, and low-migration graph-based repartitioning.

2.6 *Beyond L4: Improving Scalability with* M³ *and* SEMPEROS

We conclude the performance and scalability section of this paper with an outlook on a new kernel architecture suitable for heterogeneous many-core systems: SEMPEROS [25].

M³ SEMPEROS is based on M³ [1], a hardware/software co-designed system architecture. Like L4Re, which we described in Sect. 2.2, these systems are microkernel OSes that manage access rights to all system resources based on *capabilities*. A process can only use a resource, if it owns a capability to it. Such resources include threads, memory allocations, and files, but they are also used to grant and revoke access to CPU cores and the ability to send messages between the threads running on these cores. A key aspect of the M³ design is that the OS kernel runs on just one core, which remotely manages all other cores[2] in the system.

Heterogeneous Architectures Since it is not necessary that all cores of a system can run an OS kernel, M³ is suitable for heterogeneous system architectures with different kinds of CPU cores. The current No. 1 system in the TOP500 list of supercomputers, China's Sunway TaihuLight system [19], is based on such an architecture. Each node has "big" cores capable of running an OS kernel, as well as many small compute cores that are optimized for computation, but lack the architectural support for an OS kernel.

SEMPEROS To put hundreds of cores under the control of a microkernel OS, the kernel and its capability system need to scale. SEMPEROS extends M³ to manage the

[2] M³ provides a hardware abstraction to integrate accelerators in the same way as general-purpose cores. Therefore, we usually use the term *processing element* in an M³ context. HPC workloads can benefit from generalized accelerator support, but it does not influence how the capability system works. In this paper, we therefore use the common term *cores* to simplify the explanation.

system using multiple kernels. The compute cores of the system are organized into as many partitions as there are kernel instances, thereby increasing the total number of cores (and compute threads) the system can handle. Each of the SEMPEROS kernel instances executes on its own privileged core, but they coordinate with each other via a *distributed capability protocol* [25]. The collaboration between cores managed by different kernels is transparent to the applications.

Capability Model SEMPEROS implements partitioned capabilities. The capabilities are stored within a protected address space and the kernel supervises capability operations. Application processes can delegate and revoke capabilities via the respective system calls. The kernel records the delegations in a capability tree that stores the relations between capabilities. Using this capability tree, SEMPEROS implements recursive revocation, by which all access rights that originated from the specific capability can be revoked by deleting this capability and all its descendants.

Distributed Capability Protocol In a distributed capability system, the data structure storing the capability tree is split between multiple kernels. Because the same capabilities can be manipulated simultaneously by different kernels in the system, SEMPEROS kernel instances need to coordinate certain capability operations. We analyzed all possible interleavings of capability operations (e.g., during delegation or revocation) and developed a protocol that prevents inconsistencies in the capability tree. This protocol integrates a confirmation for capability delegation (similar to a two-way handshake) and a distributed mark-and-sweep algorithm to revoke capabilities [25].

Evaluation Setup We evaluated SEMPEROS using the cycle-accurate gem5 simulator [6]. The experiments were run with up to 640 out-of-order cores integrated into a single network-on-chip. The applications used in our evaluation stress the capability system, as they extensively use OS services. In particular, they make heavy use of the M^3/SEMPEROS in-memory file system, which grants access to memory ranges within files by delegating memory capabilities to applications and revoking those capabilities when files are closed again. We assume that similar usage patterns would also occur during checkpointing operations on a future HPC system with storage-class memory in each node.

Scalability Results To quantify the scalability of SEMPEROS, we computed the parallel efficiency, exposing the runtime overhead a single benchmark instance (i.e. process) experiences when it is executed in parallel with a number of identical benchmark instances. Figure 10a depicts an overview of the parallel efficiency of all application benchmarks we evaluated. The scalability of the applications improve when increasing the number of kernels managing the system as depicted in Fig. 10b.

The capability protocol in general is applicable to any distributed capability system implementing global IDs and stores all capability relations in a capability tree.

www.dbooks.org

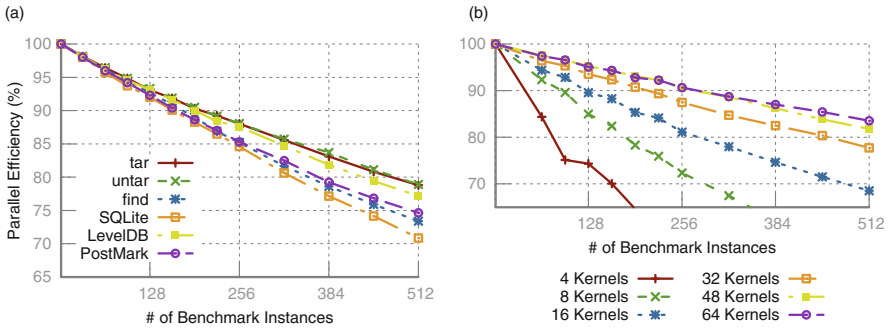


Fig. 10 Scalability evaluation of SEMPEROS. (**a**) Parallel efficiency of application benchmarks. (**b**) Level DB key-value store

3 Algorithms and System Support for Fault Tolerance

In this section we discuss FFMK research results on fault tolerance. Our activities concentrated on two areas: First, we introduce new methods for resilience of MPI applications at scale. To this end, we developed probabilistic and deterministic algorithms for resilience of collective operations (Sects. 3.1 through 3.3). Second, we present new approaches to coordinate and optimize concurrent checkpointing of multiple jobs (Sect. 3.4) and to improve multi-level checkpointing for a single parallel job (Sect. 3.5).

3.1 Resilience in MPI Collective Operations

MPI libraries are usually not fault tolerant and therefore not able to complete a collective operation correctly in case of a fault. Instead, the whole application will either hang or crash. Unfortunately, the overall system reliability decreases as the number of processes involved in the computation grows.

Existing Recovery Approaches The MPI standard does not address resilience of parallel applications, with the exception of return codes for detected errors. Currently, there are multiple outstanding proposals how to address faults during MPI run-time. Most work on this subject is characterized as either *backward recovery* or *forward recovery*. The former tries to restore a correct *past* state, whereas the latter attempts to establish a *new* correct state. For example, ULFM [8] applies forward recovery, entailing a set of tools that allow applications to deal with detected faults.

Fault-Tolerant Algorithms Fault-Tolerant Collective Operations (FTCO) [43] is a new forward recovery approach. FTCO relies on parallel algorithms that apply to tree-based collective operations in MPI. It includes resilient versions of collective operations such as *Bcast*, *Reduce* and *Allreduce*, for any tree topology. This

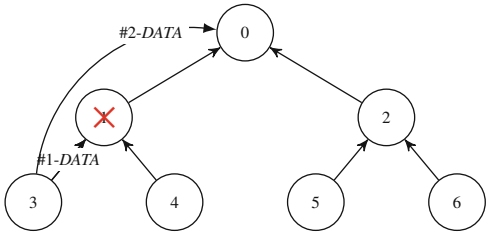


Fig. 11 Message flow after *DATA* timeout: Node #3 overcomes fault detected on node #1 by bypassing it and sending the data to the next node #0 along the tree

algorithm detects faults by a per-node calculated timeout and overcomes faults by excluding failed processes. It is intended for use-cases that can tolerate process faults, such as Monte–Carlo method or PDE solvers. FTCO delivers a result to every live process, so that the application may keep running without handling those faults. The FTCO algorithm minimizes the fault-free performance penalty and shows a small increase in latency and messages per fault, regardless of job size. This offers a transparent and scalable forward recovery alternative to the costly legacy backward recovery mechanisms. For example, Fig. 11 demonstrates a simple case, where node #3 overcomes the fault detected on node #1 by bypassing it and sending the data to the next node (#0) along the tree.

FTCO Results FTCO differs from other approaches, such as ULFM, by localizing the detection and recovery of faults, while other approaches involve the entire group of processes in the MPI job. Our experimental results with the FTCO approach support this claim, showing that the latency with FTCO is proportional to the number of serial faults, regardless of the number of MPI proceeses: Table 2 shows the FTCO latency of the *Allreduce* operation for different combinations of offline faults and increasing number of processes. We chose a timeout of 2 s, which is the default for ULFM [43]. In the table, each figure represents the average longest time (in seconds) for a process to complete the same *Allreduce* call. In a tree topology, multiple faults could be either serial or parallel: parallel faults occur in different subtrees, thus their recovery time may overlap, while serial faults are handled one after the other. We found that two parallel faults take approximately the same amount of time as a single fault; adding a third parallel fault to two serial faults does not change the latency.

Table 2 FTCO: average longest time in seconds for a process to complete *Allreduce* call

No. of process	No faults	1 fault	2 parallel faults	2 serial faults	3 mixed faults
64	0.0005	2.0095	2.0094	4.0194	4.0194
128	0.0006	2.0123	2.0123	4.0221	4.0222
256	0.0007	2.0121	2.0123	4.0220	4.0219
384	0.0017	2.0114	2.0323	4.0404	4.0217
512	0.0024	2.0343	2.0110	4.0421	4.0422

www.dbooks.org

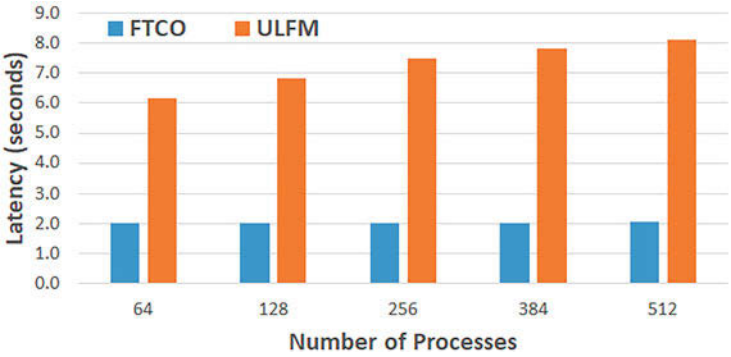


Fig. 12 The latency cost of one fault (in seconds) with FTCO and ULFM: FTCO outperforms ULFM; both approaches use the same fault-detection timeouts and tree topology

FTCO vs ULFM Figure 12 shows a comparison between the latency of FTCO and ULFM for the *Allreduce* operation with a single fault and an increasing number of processes. In both cases we used the same parameters, like fault detection timeouts and tree topology. Each test started by injecting a fault into one process and then attempted an *Allreduce* on multiple communicators with that process.

Integration of FTCO into UCX We developed a prototype library to further measure the performance of FTCO based on UCX. UCX [54] is an open-source point-to-point communication library, optimized for performance on low-latency interconnects. UCX is designed to provide a high-level of abstraction for communication, and to consider the specifics of the local NIC, to find the optimal send and receive methods. UCX queries the capabilities of each NIC to establish which hardware accelerations are present, and chooses the optimal parameters for sending messages, including the NIC, port and protocol (e.g. rendezvous). In order to apply fault-tolerance algorithms to MPI applications, we extended UCX with an implementation of MPI's collective operations. Our library is suitable for applications that can withstand partial failures, where the overall result is not effected by some faults, or where the application can overcome them.

Our extension contains a basic, deterministic implementation for fault-free collective operations. It demonstrated the benefit of a persistent collective operation: applications often repeat the same collective operation call, and so the library can reuse past structures instead of creating it from scratch for each call. Our UCX-based library can be used with any MPI implementation, and provides a foundation for further experimental results and optimizations.

3.2 Corrected Gossip Algorithms

An alternative to the deterministic approach taken with FTCO is to use randomized gossip-based algorithms, which have been shown to yield better recovery latency for some applications. Gossip algorithms have been widely successful in various contexts that did not require strong consistency. Yet, they become rapidly inefficient once about 50% of the nodes were reached, because messages are more likely to be sent to nodes that have already received a message before.

Gossip-Based Broadcast Let us consider how broadcast among processes can be implemented using gossip. The root sends the broadcast payload to a random subset of other processes. When a new process receives the payload and gets *colored*, it starts sending messages to randomly selected processes as well. This dissemination runs for a fixed period of time, after which all colored processes have received the broadcast payload.

Closing the Gaps The gossip phase attempts to color as many processes as possible, but due to potential failures and the random nature of gossip, the protocol cannot guarantee that all live processes are actually colored. Our new algorithm therefore enters a deterministic correction phase, in which it tries to color these remaining processes. For correction, all processes are reorganized into a virtual ring (e.g., according to their MPI rank numbers from 0 to $P-1$ for P processes). On this ring, uncolored processes create *gaps*, where the *maximum gap size* is the length of the longest sequence of uncolored processes. All colored processes now send messages to their neighbors on the ring, thereby closing the gaps with few messages per node.

Corrected Gossip Algorithms We designed three different protocols based on this idea of combining randomized and deterministic algorithms for improving the broadcast latency [27]. These three algorithms allow us to choose various tradeoffs between consistency, simplicity, and performance. The three algorithms are: (1) *opportunistic*, which applies the correction without checking for completion; (2) *checked*, which runs the correction until all nodes received the message, provided that no nodes fail during the correction; and (3) *fail-proof*, which applies the correction and guarantees that all active nodes receive the message, provided that no more than f nodes fail during the operation. Our algorithms do not require multicast, failure detectors, timeouts, acknowledgments, or reconfiguration procedures. The result of this work is the "corrected-gossip" paradigm [27].

Framework for Collective Algorithms Based on the corrected-gossip paradigm, we also developed a framework for failure-proof collective operations that generates online an independent spanning tree. Generation can be completed successfully even with an arbitrary number of active nodes and up to f online failures. Based on the system's mean time between failures (MTBF), an appropriate value f could be chosen as the maximal number of faults supported by the algorithm. Compared to alternative methods for fault recovery, this approach allows a trivial

www.dbooks.org

recovery procedure, provided that sufficient spanning trees are maintained during the application run.

3.3 Corrected Tree Algorithms

The reliability properties of all correction schemes of "corrected gossip" have been proven [27]. However, due to being probabilistic, the gossip algorithm used in the dissemination phase needs to send more messages than an optimal tree-based broadcast algorithm in order to color nodes. Tree-based dissemination will, however, miss a large number of processes, if any process close to the tree's root fails. In general, failure of any non-leaf process in the tree results in all its descendants remaining uncolored.

Can We Save the Trees? We developed a *corrected tree* algorithm which splits communication among processes into two phases, exactly like we did with corrected gossip: *dissemination* and *correction*. The idea is to correct the result of a failed tree-based broadcast during the dissemination phase. With failure-proof correction, full coloring can be guaranteed even if processes fail during the broadcast [27].

Requirements for Corrected Trees The goal of the broadcast is to guarantee information propagation from the root process to every live process, even if some processes fail. In a broadcast operation among P processes, the *root* process propagates a message reliably to all other processes. Without loss of generality we assume the root process to have rank 0, other processes have ranks $1, \ldots, P-1$. To ensure that correction can color uncolored processes quickly, the maximum gap size ought to be small. A tree maintaining such a property should have its subtrees spread across the correction ring to avoid the danger of having uncolored processes cluster together on the ring. For lowering correction latency, multiple small gaps are better than few large ones.

Interleaved Trees The key idea behind corrected tree algorithms is that nodes of the tree can be renumbered in such a way that parent and child nodes do not become neighbors on the ring. Instead, nodes from a subtree below a failed node shall always have a close neighbor from outside their own subtree, so the they can be colored in the correction phase. In this paper, we only describe how to interleave k-ary trees. However, the scheme is also available for Lamé trees, which include Binomial trees. A more detailed description is given in the original publication [31].

Interleaving k-ary Trees Given the root process at level 0, a full k-ary tree has k^ℓ processes at level ℓ, and k^ℓ subtrees that have their root process at that level. The processes in these subtrees can have a distance of k^ℓ in the ring. Process r has the child processes r':

$$\{r' \mid r' = r + i \cdot k^\ell, 0 < i \leq k, r' < P\}$$

This interleaving ensures that a failing process on level ℓ leads to every k^ℓ-th process being uncolored. Thus, $f = k^\ell - 1$ failures on level ℓ or below can be tolerated, and still every k^ℓ-th process will be colored after the dissemination.

Correction Phase The correction phase follows the dissemination phase and is independent of the tree type. It ensures coloring of the processes that the dissemination phase left uncolored due to failures. All three correction algorithms developed for corrected gossip [27] are directly applicable.

Simulation Results In our simulation-based evaluation, we used the same fail-stop fault model as in the corrected-gossip publication [27]. During broadcast, every process is either dead or alive. A process either sends all messages required by the protocol or none at all, but failures can occur anywhere outside of the broadcast operation. The simulation is based on the LogP-model [10]. The graph in Fig. 13 shows the number of messages sent for different numbers of faults. All tree-based broadcasts send fewer messages than corrected gossip and are thus more efficient with regard to this metric.

Latency Measurements To measure the average latency of a broadcast, we implemented all algorithms using MPI and ran them on the Piz Daint supercomputer installed at ETH Zurich. The results shown in Fig. 14 therefore include overhead due to the physical properties of the interconnect. Note that there are no faults in this experiment, so we can compare our implementations with the broadcast implementation from Cray. Since our MPI-based implementation cannot use the shared-memory optimization the Cray algorithm uses, we include performance

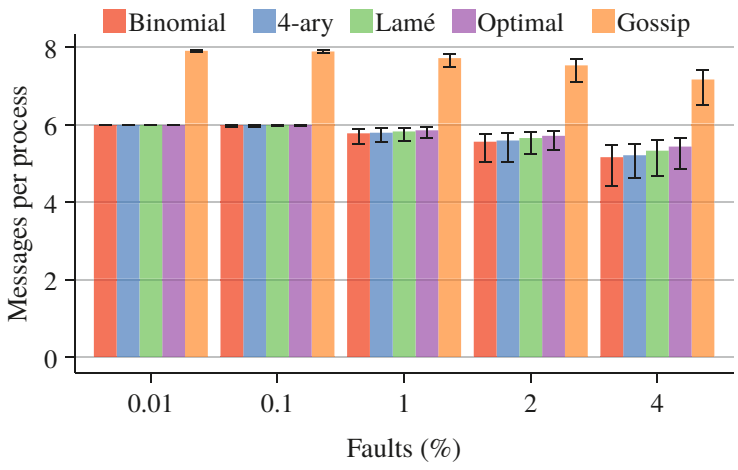


Fig. 13 Average number of messages sent per process in presence of failures; four different corrected tree algorithms and corrected gossip were simulated with varying percentage of failed processes

www.dbooks.org

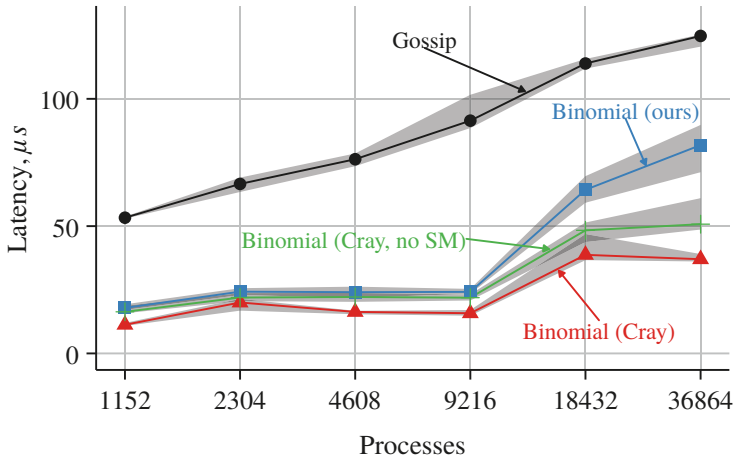


Fig. 14 Broadcast median latency of corrected tree, corrected gossip, and vendor-provided implementations of broadcast without any failures

of the Cray algorithm with shared memory disabled for reference (green line in diagram).

We find *Corrected Trees* to be a simple, yet efficient protocol for fault-tolerant collective group communication. A more detailed evaluation can be found in the publication [31].

3.4 Checkpointing Scheduling

Only some applications can benefit from forward recovery by tolerating process or node failures. All other HPC codes rely on system-level or application-assisted fault-tolerance services. Periodic checkpoint/restart (C/R) [11, 65] is an effective mechanism to alleviate failures during execution of HPC applications, which often require a vast number of nodes for a relatively long time [15]. The overhead of checkpointing should be kept as small as possible, as typically the computation has to be paused during a checkpoint write to store a consistent state of the application. Furthermore, on a large supercomputer, which is shared by several parallel jobs, checkpointing should be coordinated between applications to avoid performance bottlenecks. Otherwise, as the storage is typically shared between all applications, concurrent checkpoint writes would unnecessarily delay the computation tasks.

Uncoordinated Checkpointing Hurts As part of the FFMK project, we designed a mechanism to coordinate concurrent checkpoints of large applications running on a supercomputer and sharing a dedicated burst-buffer [42] or parallel file system (PFS) [41] for checkpointing [22]. We assume that the system executes multiple

parallel applications at the same time and that it provides a single shared PFS (e.g., Lustre, GPFS, ...) for all applications. Typically, each application uses fixed checkpoint intervals calculated independently of the other applications based on Daly's method [11]. However, Daly assumes a constant checkpoint write duration during the application's lifetime, whereas in case of concurrent checkpointing of different applications checkpoint costs vary and depend on other applications' activities due to limited write bandwidth and other interferences.

Checkpoint Scheduling Approach In our work, we take this variation into account and rearrange the checkpoints of jobs, by predicting conflicts, in order to minimize the overall system resource usage. This is achieved by *scheduling checkpoints* earlier or later than originally planned. The decision whether to postpone a checkpoint to another timeslot is based on a constructed optimization problem for the predicted conflicts. By minimizing the constructed problem, non-overlapping timeslots to write checkpoints are found and applied.

Checkpoint Scheduling Architecture While system-level checkpointing takes advantage of system-wide information not visible to jobs such as mean-time to failure (MTTF), bandwidth, or other jobs' activities, it suffers from unnecessarily large checkpoints (higher checkpoint costs). On the other hand, user-defined checkpointing is done within the application which uses a runtime library for checkpointing. The library remembers the important data to be saved and also knows at which time checkpoints should be written (application-level information). This way the approach reduces the checkpoint size while lacking the system-wide information. In our work, we combined system-level with user-defined checkpointing in order to leverage the advantages of both methods [22]. The coordination among different jobs is performed by a coordinator service running on the system side. The service has access to system-wide information like MTTF, the bandwidth of shared storage systems, and the activities of other jobs (system-level checkpointing).

User-Level Library Support To achieve this, we modified the SCR library [44] to provide it with the ability to interact with our coordinator service, which instructs SCR when to perform checkpoint requests. The application uses the library to write checkpoints (user-level checkpointing). This library cooperates with the coordinator-service, whereby whenever a checkpoint call is made by the application (which is done frequently and periodically), a request is sent to the coordinator and if accepted, the write will be performed by the library. Otherwise, the checkpoint call will be ignored and the application continues with the computation.

System-Level Coordinator The decisions on the coordinator side are based on solved optimization problems constructed for the predicted conflicts. Then the incoming checkpoint requests are either accepted or rejected based on the computed decisions to get optimal checkpoint times.

Figure 15 shows that we are able to reduce the C/R overhead by up to 20% by coordinating checkpoints compared to state-of-the-art approaches.

www.dbooks.org

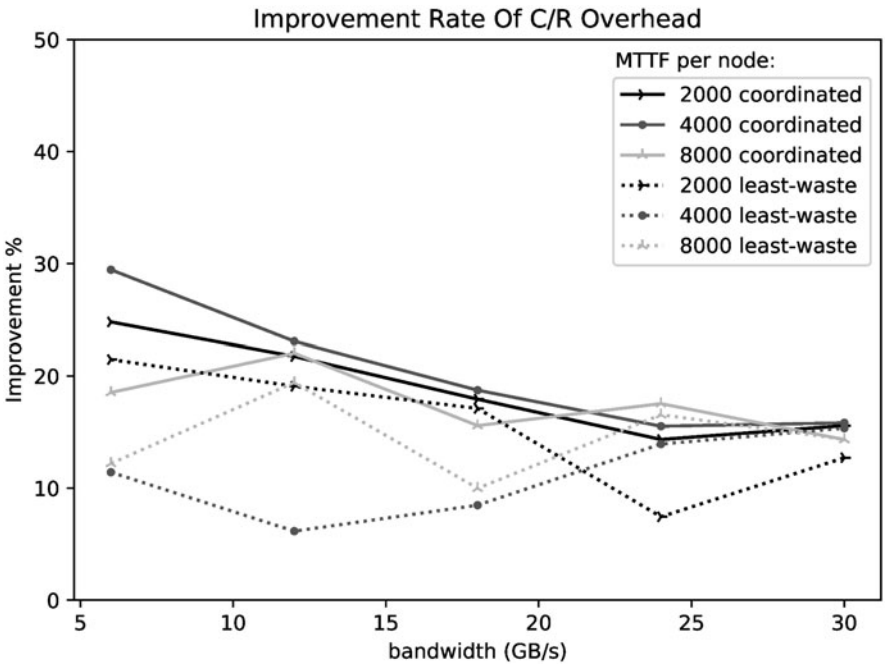


Fig. 15 Improvement rate of C/R overhead

3.5 Multi-Level Checkpoint/Restart

To further strengthen the C/R mechanism of the FFMK OS platform, *multi-level checkpoint/restart* is used as a solution to address a wide range of failures of different severity occurring in large supercomputers [21]. Severe failures require application state to be recovered from the most stable storage devices. However, this comes at high cost, as stable storage devices are slower to write to than transient storage. This is the key motivation of using multi-level checkpoint/restart.

Component Failure Rates To fully grasp the multi-level hierarchy of failures and storage devices, we constructed a comprehensive model of a typical large cluster containing nodes, local storages (ramdisk, SSD), network switches, power supplies, different shared storage systems (burst-buffer, PFS), etc. Additionally, we modeled 5 different levels of checkpoints, all of which are supported by the SCR [44] library and most of them are also available on other checkpointing libraries as well (FTI [4], VeloC [46]):

- **Local Level:** The first checkpoint level is the node's local storage (ramdisk, SSD). While this level benefits from the lowest checkpointing cost among all levels, it cannot survive fatal faults where the node becomes unavailable.

- **XOR Level:** The next level is the XOR level, at which for each node a parity segment is computed and distributed among a set of nodes (XOR group) according to [24, 47]. Each node stores its local checkpoint along with the XOR parity data. This level survives a single node of an XOR group becoming unavailable.
- **Partner Level:** A more stable level is the partner scheme, where the checkpoint is stored in the node's local storage along with the partner node's storage (two copies). This level fails to restart the job, if a node and its partner fail together.
- **Shared Levels:** The most stable levels use shared storage (among all compute nodes), namely the burst buffer and the parallel file system. However, they provide lower checkpointing bandwidth per node, because many (or all) nodes may access them concurrently.

Modeling Failures Per Component To have a proper estimation of the stability of different checkpoint levels (i.e., how often they fail), we studied a wide range of possible failures occurring on supercomputers and investigated their effect on each checkpoint level. Using the estimations, we defined the optimum checkpoint intervals for each level minimizing the application's lifetime. To study different types of failures, we differentiated light faults and fatal failures, where after a light fault the node stays alive and the job can be restarted from the local storage device. A fatal failure makes a node or a set of nodes unavailable (node down). In this case, the job must be restarted from more stable levels (XOR, partner, etc.). Additionally, we took correlations into account by arranging the nodes of a cluster in different correlated groups (network, power supply, etc.), whereby the nodes belonging to the same correlated group are vulnerable to failures of the same origin making all nodes in the group unavailable at the same time (e.g., a network switch outage). Figure 16 visualizes, how this correlation of nodes translates into a correlation graph.

In addition to the compute nodes, we considered I/O nodes (burst-buffer) which are also vulnerable to failures making the burst-buffer unavailable for a restart. Although such faults do not interrupt the job's computation, losing checkpoints on the burst-buffer may impact the job's lifetime. Another case of a checkpoint's

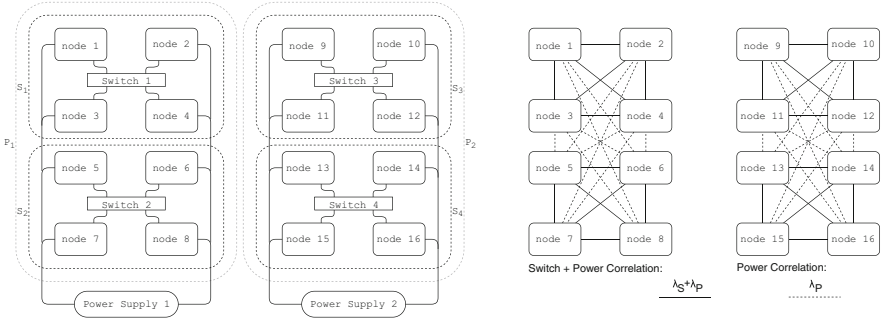


Fig. 16 Correlation of nodes in a cluster

www.dbooks.org

unavailability occurs, if due to a node down, the job is restarted from the XOR level and the new node allocated to the job lacks the partner level checkpoint. Hence, in case of another failure, an attempt to restore from partner-level will fail to recover the just restarted job a second time until the next partner checkpoint is written.

Model-Based Adaptation of Checkpoint Behavior Finally, we introduced a mechanism to model the expected lifetime of applications writing multi-level checkpoints on a given set of checkpoint levels considering all investigated types of failures. Using this model, we properly choose the optimal set of checkpoint levels for each job minimizing the expected lifetime.

Multi-Level Checkpoint Architecture Our implementation of multi-level checkpoint/restart consists of a system-level daemon and a library that is linked into the application. The daemon communicates with the library in the application using TCP sockets and directs the local checkpoints (written by the application) to different checkpoint levels transparent to the application. It computes the optimum checkpoint intervals of each level using system-wide information and application-specific parameters. For each checkpoint call made through the library, the daemon determines the optimal level of the checkpoint (or simply rejects it). We employ the following optimizations and automations in our design to further improve performance and reliability:

- **Asynchronous Checkpointing:** To further reduce the checkpointing costs of the lower levels, checkpoint writes are performed asynchronously. The application writes its state to the node's local storage, providing fast write performance. Thereafter, the operation of checkpointing at the lower levels (e.g., partner, burst buffer, PFS) is performed by the daemon in the background while the application returns to the computation. During the background operation of the daemon, every incoming request for checkpointing to the same level from the job is ignored until the persisting is finished.
- **Fast Recovery:** To ensure the fast recovery in case of light failures (node alive), ULFM [7] is used to provide the capability of automatically recovering the failed ranks with the most recent checkpoint and reordering the ranks to the original arrangement, transparent to the developer, application, and the resource manager.
- **Job Life Cycle and Recovery** When a new job is started, a short-lived controller service is executed by the first rank on each node (i.e., the node's local rank 0). This service forks a daemon process for each rank on the node, connects them to the corresponding ranks, and then terminates. The per-rank daemons are responsible for sending status information and receiving instructions. The per-rank daemons stay alive until the application is finished with the computation. If a rank fails, its daemon terminates and the rank is later restarted using ULFM. When a rank is restarted, or when it notices that its daemon died, it will execute the previously mentioned controller service again, which will then reconnect a new daemon process with the rank. This is done transparently to the application and the existence of both daemons and controller services remains transparent to the user and the developer.

- **Rotating Partner Nodes:** To further enhance the stability of the partner level, a novel approach will be engaged in which there are no fixed partners for the nodes, instead, the partner node of a specific node is changed on each turn and chosen by the daemon. This allows reducing the probability of unavailability of checkpoints at this level. In addition, partner checkpoints and data transfers will be performed by service daemons of both nodes using Remote Direct Memory Access (RDMA) in order to have fast partner checkpoints in the background.

Our evaluations of the introduced multi-level checkpoint/restart model (the failure rates and optimal intervals) show that we managed to reduce the C/R overhead of jobs by up to 10% in the investigated cases (50,000 nodes, 3–5 levels, 6–240 GB checkpoints) compared to the state-of-the-art approach. Our observations indicate that the overhead is reduced further as the number of levels or the checkpoint size increases.

4 Conclusions

The FFMK project had the very ambitious goal of creating a new operating-system (OS) platform for Exascale computing. Unfortunately, we did not reach the milestone of a fully integrated prototype at the end of phase 2. However, several essential building blocks have been adapted for HPC and new algorithms and services have been developed to meet the challenges we expect from these upcoming extreme-scale systems.

Multi-Kernel Node OS At the node level, the mature L4Re microkernel together with L^4Linux enable the decoupled-thread model, which combines noise-free execution and full Linux compatibility for applications. We studied and quantified the influence of hardware performance variation, a source of noise that still remained. Furthermore, we gave an outlook on how to manage OS-level resources at extreme scales of parallelism with a kernel architecture for scalable capability management.

Dynamic Platform Management Although mostly an activity from the first three-year phase of the project, we continued research on dynamic platform management at the system level. We researched fault-tolerant gossip-based algorithms for obtaining load and health information from nodes. We integrated these information dissemination algorithms with a per-node management daemon and a distributed decision making module. However, one component of the FFMK architecture's load balancing service remained elusive: migration of processes has been shown to work at small scale for simple InfiniBand-based programs, but robust C/R-based migration is not available yet. In Phase 2, we developed highly efficient diffusion-based load balancing algorithms; they complement the architecture and can already be used for application-level load balancing, as they have been integrated into a widely-used framework.

www.dbooks.org

Fault Tolerance The main activities of the second phase of this project concentrated on efficient and fault-tolerant communication, as well as system-level support for reducing checkpoint overhead. Fault-tolerant collective operations (FTCO) have been developed as a drop-in replacement for MPI libraries and were prototyped in the UCX communication library. As a potentially more efficient alternative, we introduced a new class of two-step algorithms based on either gossip or trees: Corrected Gossip and Corrected Trees enable inherently fault-tolerant collective operations for applications that can continue even after some processes failed. Globally optimized checkpoint scheduling and multi-level checkpointing minimize checkpoint/restart overhead for all other applications. The latter optimizes the cost of checkpointing based on a model that includes expected failure rates of components, a failure-correlation graph, and available write bandwidth across the storage hierarchy.

Together, all newly developed algorithms, system services, and low-level OS components form the basis of a fast, scalable, and fault-tolerant HPC operating system for Exascale computing.

Acknowledgments This research and the work presented in this paper is supported by the German priority program 1648 "Software for Exascale Computing" via the research project FFMK [17]. We also thank the cluster of excellence "Center for Advancing Electronics Dresden" (*cfaed*). The authors acknowledge the Jülich Supercomputing Centre, the Gauss Centre for Supercomputing, the John von Neumann Institute for Computing, and the Swiss National Supercomputing Centre (CSCS) for providing compute time on their supercomputer systems.

References

1. Asmussen, N., Völp, M., Nöthen, B., Härtig, H., Fettweis, G.: M3: A hardware/operating-system co-design to tame heterogeneous manycores. In: Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (2016)
2. Barak, A., Guday, S., Wheeler, R.: The MOSIX Distributed Operating System: Load Balancing for UNIX. Lecture Notes in Computer Science, vol. 672. Springer, Berlin (1993)
3. Barak, A., Drezner, Z., Levy, E., Lieber, M., Shiloh, A.: Resilient gossip algorithms for collecting online management information in exascale clusters. Concurr. Comput. Pract. Exp. **27**(17), 4797–4818 (2015)
4. Bautista-Gomez, L.A., et al.: FTI: high performance fault tolerance interface for hybrid systems. In: SC'11 Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 32:1–32:32 (2011). http://doi.acm.org/10.1145/2063384.2063427
5. Beckman, P., Iskra, K., Yoshii, K., Coghlan, S.: The influence of operating systems on the performance of collective operations at extreme scale. In: 2006 IEEE International Conference on Cluster Computing, pp. 1–12 (2006). https://doi.org/10.1109/CLUSTR.2006.311846
6. Binkert, N., Beckmann, B., Black, G., Reinhardt, S.K., Saidi, A., Basu, A., Hestness, J., Hower, D.R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M.D., Wood, D.A.: The Gem5 simulator. SIGARCH Computer Architecture News (2011)
7. Bland, W.: User level failure mitigation in MPI. In: Euro-Par 2012: Parallel Processing Workshops - BDMC, CGWS, HeteroPar, HiBB, OMHI, Paraphrase, PROPER, Resilience,

UCHPC, VHPC, Rhodes Islands, August 27–31, 2012. Revised Selected Papers, pp. 499–504. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-36949-0_57
8. Bland, W., Bouteiller, A., Herault, T., Hursey, J., Bosilca, G., Dongarra, J.J.: An evaluation of user-level failure mitigation support in MPI. In: Träff, J.L., Benkner, S., Dongarra, J.J. (eds.) Recent Advances in the Message Passing Interface, pp. 193–203. Springer, Berlin (2012)
9. Cavium: ThunderX_CP Family of Workload Optimized Compute Processors (2014). https://www.marvell.com/content/dam/marvell/en/public-collateral/server-processors/marvell-server-processors-thunderx-cp-product-brief.pdf
10. Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., Von Eicken, T.: LogP: towards a realistic model of parallel computation. In: Symposium on Principles and Practice of Parallel Programming, PPoPP, pp. 1–12. ACM, New York (1993). https://doi.org/10.1145/155332.155333
11. Daly, J.T.: A higher order estimate of the optimum checkpoint interval for restart dumps. Future Gener. Comput. Syst. **22**(3), 303–312 (2006). https://doi.org/10.1016/j.future.2004.11.016
12. Devine, K., Boman, E., Heaphy, R., Hendrickson, B., Vaughan, C.: Zoltan data management services for parallel dynamic applications. Comput. Sci. Eng. **4**(2), 90–97 (2002)
13. Diekmann, R., Frommer, A., Monien, B.: Efficient schemes for nearest neighbor load balancing. Parallel Comput. **25**(7), 789–812 (1999)
14. Diekmann, R., Preis, R., Schlimbach, F., Walshaw, C.: Shape-optimized mesh partitioning and load balancing for parallel adaptive FEM. Parallel Comput. **26**(12), 1555–1581 (2000)
15. Feinberg, A.: An 83,000-processor supercomputer can only match 1% of your brain (2013). http://gizmodo.com/an-83-000-processor-supercomputer-only-matched-one-perc-1045026757
16. Ferreira, K.B., Bridges, P., Brightwell, R.: Characterizing application sensitivity to OS interference using Kernel-level noise injection. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC'08, pp. 19:1–19:12. IEEE Press, Piscataway (2008). http://dl.acm.org/citation.cfm?id=1413370.1413390
17. FFMK Website. http://ffmk.tudos.org. Accessed 5 Aug 2019
18. Forum, M.P.I.: MPI: a message-passing interface standard. Standard 3.1, University of Tennessee, Knoxville (2015)
19. Fu, H., Liao, J., Yang, J., Wang, L., Song, Z., Huang, X., Yang, C., Xue, W., Liu, F., Qiao, F., Zhao, W., Yin, X., Hou, C., Zhang, C., Ge, W., Zhang, J., Wang, Y., Zhou, C., Yang, G.: The Sunway TaihuLight supercomputer: system and applications. Sci. China Inf. Sci. **59**(7), 072001 (2016). https://doi.org/10.1007/s11432-016-5588-7
20. Gerofi, B., Takagi, M., Hori, A., Nakamura, G., Shirasawa, T., Ishikawa, Y.: On the scalability, performance isolation and device driver transparency of the IHK/McKernel hybrid lightweight kernel. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 1041–1050 (2016). https://doi.org/10.1109/IPDPS.2016.80
21. Gholami, M., Schintke, F.: Multilevel checkpoint/restart for large computational jobs on distributed computing resources. In: 38th Symposium on Reliable Distributed Systems (SRDS'19) (2019)
22. Gholami, M., Schintke, F., Schütt, T.: Checkpoint scheduling for shared usage of burst-buffers in supercomputers. In: The 47th International Conference on Parallel Processing, ICPP 2018, Workshop Proceedings, Eugene, August 13–16, 2018, pp. 44:1–44:10. ACM, New York (2018). https://doi.org/10.1145/3229710.3229755
23. Giampapa, M., Gooding, T., Inglett, T., Wisniewski, R.W.: Experiences with a lightweight supercomputer Kernel: lessons learned from Blue Gene's CNK. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC (2010). https://doi.org/10.1109/SC.2010.22
24. Gropp, W.D., et al.: Providing efficient I/O redundancy in MPI environments. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting. Lecture Notes in Computer Science, vol. 3241, pp. 77–86 (2004). https://doi.org/10.1007/978-3-540-30218-6_17

www.dbooks.org

25. Hille, M., Asmussen, N., Bhatotia, P., Härtig, H.: SemperOS: A distributed capability system. In: 2019 USENIX Annual Technical Conference (ATC) (2019)
26. Hoefler, T., Schneider, T., Lumsdaine, A.: Characterizing the influence of system noise on large-scale applications by simulation. In: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC'10. IEEE Computer Society, Washington (2010). https://doi.org/10.1109/SC.2010.12
27. Hoefler, T., Barak, A., Shiloh, A., Drezner, Z.: Corrected gossip algorithms for fast reliable broadcast on unreliable systems. In: International Parallel and Distributed Processing Symposium, IPDPS, pp. 357–366. IEEE Computer Society, Washington (2017). https://doi.org/10.1109/IPDPS.2017.36
28. IBM: Design of the IBM Blue Gene/Q Compute chip. IBM J. Res. Develop. **57**(1/2), 1:1–1:13 (2013). https://doi.org/10.1147/JRD.2012.2222991
29. Intel: Intel xeon processor E5-1600/E5-2600/E5-4600 v2 product families (2014). https://www.intel.com/content/www/us/en/processors/xeon/xeon-e5-1600-2600-vol-2-datasheet.html
30. Kelly, S.M., Brightwell, R.: Software architecture of the light weight kernel, Catamount. In: Cray User Group, pp. 16–19 (2005)
31. Küttler, M., Planeta, M., Bierbaum, J., Weinhold, C., Hä rtig, H., Barak, A., Hoefler, T.: Corrected trees for reliable group communication. In: Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, PPoPP'19, pp. 287–299. ACM, New York (2019). http://doi.acm.org/10.1145/3293883.3295721
32. Lackorzynski, A., Weinhold, C., Härtig, H.: Combining predictable execution with full-featured commodity systems. In: Proceedings of OSPERT2016, the 12th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2016, pp. 31–36 (2016)
33. Lackorzynski, A., Weinhold, C., Härtig, H.: Decoupled: Low-effort noise-free execution on commodity system. In: Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers, ROSS'16. ACM, New York (2016)
34. Lackorzynski, A., Weinhold, C., Härtig, H.: Predictable low-latency interrupt response with general-purpose systems. In: Proceedings of OSPERT2017, the 13th Annual Workshop on Operating Systems Platforms for Embedded Real-Time Applications, OSPERT 2017, pp. 19–24 (2017)
35. Lawrence Livermore National Laboratory: The FTQ/FWQ benchmark. https://asc.llnl.gov/sequoia/benchmarks/FTQ_summary_v1.1.pdf
36. Levy, E., Barak, A., Shiloh, A., Lieber, M., Weinhold, C., Härtig, H.: Overhead of a decentralized gossip algorithm on the performance of HPC applications. In: Proceedings of ROSS'14, pp. 10:1–10:7. ACM, New York (2014)
37. Lieber, M., Nagel, W.E.: Highly scalable sfc-based dynamic load balancing and its application to atmospheric modeling. Future Gener. Comput. Syst. **82**, 575–590 (2018)
38. Lieber, M., Grützun, V., Wolke, R., Müller, M.S., Nagel, W.E.: Highly scalable dynamic load balancing in the atmospheric modeling system COSMO-SPECS+FD4. In: International Workshop on Applied Parallel Computing PARA 2010: Applied Parallel and Scientific Computing 2010. Lecture Notes in Computer Science, vol. 7133, pp. 131–141. Springer, Berlin (2012)
39. Lieber, M., Gößner, K., Nagel, W.E.: The potential of diffusive load balancing at large scale. In: Proceedings of the 23rd European MPI Users' Group Meeting (EuroMPI 2016), pp. 154–157 (2016)
40. Liedtke, J.: On micro-kernel construction. In: SOSP'95: Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles, pp. 237–250. ACM Press, New York (1995). http://doi.acm.org/10.1145/224056.224075
41. Ligon, W.B., Ross, R.B.: Implementation and performance of a parallel file system for high performance distributed applications. In: Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing (HPDC), pp. 471–480 (1996). https://doi.org/10.1109/HPDC.1996.546218

42. Liu, N., et al.: On the role of burst buffers in leadership-class storage systems. In: Proceedings of the 2012 IEEE Conference on Massive Data Storage (MSST), pp. 1–11 (2012). https://doi.org/10.1109/MSST.2012.6232369
43. Margolin, A., Barak, A.: Tree-based fault-tolerant collective operations for MPI. In: Workshop on Exascale MPI (ExaMPI) (2018)
44. Moody, A., Bronevetsky, G., Mohror, K., de Supinski, B.R.: Design, modeling, and evaluation of a scalable multi-level checkpointing system. In: 2010 International Conference for High Performance Computing, Networking, Storage and Analysis (SC) pp. 1–11 (2010). https://doi.org/10.1109/SC.2010.18
45. Muthukrishnan, S., Ghosh, B., Schultz, M.H.: First and second order diffusive methods for rapid, coarse, distributed load balancing. Theory Comput. Syst. **31**, 331–354 (1998)
46. Nicolae, B., et al.: Veloc: Very low overhead checkpointing system. https://veloc.readthedocs.io/en/latest/
47. Patterson, D.A., et al.: A case for redundant arrays of inexpensive disks (RAID). In: ACM SIGMOD Record, pp. 109–116 (1988). http://doi.acm.org/10.1145/50202.50214
48. Pedretti, K.T., Levenhagen, M., Ferreira, K., Brightwell, R., Kelly, S., Bridges, P., Hudson, T.: LDRD final report: a lightweight operating system for multi-core capability class supercomputers. Technical report SAND2010-6232, Sandia National Laboratories (2010)
49. Petrini, F., Kerbyson, D., Pakin, S.: The case of the missing supercomputer performance: achieving optimal performance on the 8,192 processors of ASCI Q. In: Proceedings of the 15th Annual IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Anaylsis (SC'03) (2003)
50. Riesen, R., Brightwell, R., Bridges, P.G., Hudson, T., Maccabe, A.B., Widener, P.M., Ferreira, K.: Designing and implementing lightweight kernels for capability computing. Concurrency and Computation: Practice and Experience **21**(6), 793–817 (2009). http://dx.doi.org/10.1002/cpe.v21:6
51. Riesen, R., Maccabe, A.B., Gerofi, B., Lombard, D.N., Lange, J.J., Pedretti, K., Ferreira, K., Lang, M., Keppel, P., Wisniewski, R.W., Brightwell, R., Inglett, T., Park, Y., Ishikawa, Y.: What is a lightweight kernel? In: Proceedings of the 5th International Workshop on Runtime and Operating Systems for Supercomputers, ROSS. ACM, New York (2015). https://doi.org/10.1145/2768405.2768414
52. Schloegel, K., Karypis, G., Kumar, V.: A unified algorithm for load-balancing adaptive scientific simulations. In: Proceedings of the IEEE/ACM SC2000 Conference, pp. 59–59 (2000)
53. Seelam, S., Fong, L., Tantawi, A., Lewars, J., Divirgilio, J., Gildea, K.: Extreme scale computing: modeling the impact of system noise in multicore clustered systems. In: 2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS) (2010). https://doi.org/10.1109/IPDPS.2010.5470398
54. Shamis, P., Venkata, M.G., Lopez, M.G., Baker, M.B., Hernandez, O., Itigin, Y., Dubman, M., Shainer, G., Graham, R.L., Liss, L., Shahar, Y., Potluri, S., Rossetti, D., Becker, D., Poole, D., Lamb, C., Kumar, S., Stunkel, C., Bosilca, G., Bouteiller, A.: UCX: an open source framework for HPC network APIs and beyond. In: 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, pp. 40–43 (2015)
55. Shimosawa, T., Gerofi, B., Takagi, M., Nakamura, G., Shirasawa, T., Saeki, Y., Shimizu, M., Hori, A., Ishikawa, Y.: Interface for Heterogeneous Kernels: a framework to enable hybrid OS designs targeting high performance computing on manycore architectures. In: 21th International Conference on High Performance Computing, HiPC (2014)
56. Sodani, A.: Knights landing (KNL): 2nd generation intel xeon phi processor. In: 2015 IEEE Hot Chips 27 Symposium (HCS), pp. 1–24 (2015). https://doi.org/10.1109/HOTCHIPS.2015.7477467
57. Teresco, J.D., Devine, K.D., Flaherty, J.E.: Partitioning and dynamic load balancing for the numerical solution of partial differential equations. In: Numerical Solution of Partial Differential Equations on Parallel Computers. Lecture Notes in Computational Science and Engineering, vol. 51, pp. 55–88. Springer, Berlin (2006)

www.dbooks.org

58. Walshaw, C., Cross, M.: Jostle – multilevel graph partitioning software: an overview. In: Mesh Partitioning Techniques and Domain Decomposition Methods, chap. 2, pp. 27–58 (2007)
59. Weinhold, C., Lackorzynski, A., Bierbaum, J., Küttler, M., Planeta, M., Härtig, H., Shiloh, A., Levy, E., Ben-Nun, T., Barak, A., Steinke, T., Schütt, T., Fajerski, J., Reinefeld, A., Lieber, M., Nagel, W.E.: FFMK: a fast and fault-tolerant microkernel-based system for exascale computing. In: Bungartz, H.J., Neumann, P., Nagel, W.E. (eds.) Software for Exascale Computing - SPPEXA 2013–2015, pp. 405–426. Springer, Cham (2016)
60. Weinhold, C., Lackorzynski, A., Härtig, H.: FFMK: an HPC OS based on the L4Re Microkernel. In: R.W. Wisniewski, B. Gerofi, R. Riesen, Y. Ishikawa (eds.) Operating Systems for Supercomputers and High Performance Computing. Springer Singapore (2019)
61. Weisbach, H., Gerofi, B., Kocoloski, B., Härtig, H., Ishikawa, Y.: Hardware performance variation: a comparative study using lightweight kernels. In: Yokota, R., Weiland, M., Keyes, D., Trinitis, C. (eds.) High Performance Computing, pp. 246–265. Springer, Cham (2018)
62. Wende, F., Steinke, T., Reinefeld, A.: The impact of process placement and oversubscription on application performance: a case study for exascale computing. In: Gray, A., Smith, L., Weiland, M. (eds.) Proceedings of the 3rd International Conference on Exascale Applications and Software, EASC 2015, pp. 13–18 (2015)
63. Wisniewski, R.W., Inglett, T., Keppel, P., Murty, R., Riesen, R.: mOS: an architecture for extreme-scale operating systems. In: Proceedings of the 4th International Workshop on Runtime and Operating Systems for Supercomputers (ROSS'14), pp. 2:1–2:8. ACM, New York (2014)
64. Yoshida, T., Hondou, M., Tabata, T., Kan, R., Kiyota, N., Kojima, H., Hosoe, K., Okano, H.: Sparc64 XIfx: Fujitsu's next-generation processor for high-performance computing. IEEE Micro **35**(2), 6–14 (2015). https://doi.org/10.1109/MM.2015.11
65. Young, J.W.: A first order approximation to the optimal checkpoint interval. Commun. ACM **17**(9), 530–531 (1974). http://doi.acm.org/10.1145/361147.361115

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



GROMEX: A Scalable and Versatile Fast Multipole Method for Biomolecular Simulation



Bartosz Kohnke, Thomas R. Ullmann, Andreas Beckmann, Ivo Kabadshow, David Haensel, Laura Morgenstern, Plamen Dobrev, Gerrit Groenhof, Carsten Kutzner, Berk Hess, Holger Dachsel, and Helmut Grubmüller

Abstract Atomistic simulations of large biomolecular systems with chemical variability such as constant pH dynamic protonation offer multiple challenges in high performance computing. One of them is the correct treatment of the involved electrostatics in an efficient and highly scalable way. Here we review and assess two of the main building blocks that will permit such simulations: (1) An electrostatics library based on the Fast Multipole Method (FMM) that treats local alternative charge distributions with minimal overhead, and (2) A λ-dynamics module working in tandem with the FMM that enables various types of chemical transitions during the simulation. Our λ-dynamics and FMM implementations do not rely on third-party libraries but are exclusively using C++ language features and they are tailored to the specific requirements of molecular dynamics simulation suites such as GROMACS. The FMM library supports fractional tree depths and allows for rigorous error control and automatic performance optimization at runtime. Near-optimal performance is achieved on various SIMD architectures and on GPUs using CUDA. For exascale systems, we expect our approach to outperform current implementations based on Particle Mesh Ewald (PME) electrostatics, because FMM avoids the communication bottlenecks caused by the parallel fast Fourier transformations needed for PME.

B. Kohnke · T. R. Ullmann · P. Dobrev · C. Kutzner (✉) · H. Grubmüller
Max Planck Institute for Biophysical Chemistry, Göttingen, Germany
e-mail: hgrubmu@gwdg.de

A. Beckmann · I. Kabadshow · D. Haensel · L. Morgenstern · H. Dachsel
Forschungszentrum Jülich, Jülich, Germany
e-mail: h.dachsel@fz-juelich.de

G. Groenhof
University of Jyväskylä, Jyväskylä, Finland
e-mail: gerrit.x.groenhof@jyu.fi

B. Hess
Science for Life Laboratory, KTH Royal Institute of Technology, Stockholm, Sweden
e-mail: hess@kth.se

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_17

www.dbooks.org

1 Introduction

The majority of cellular function is carried out by biological nanomachines made of proteins. Ranging from transporters to enzymes, from motor to signalling proteins, conformational transitions are frequently at the core of protein function, which renders the detailed understanding of the involved dynamics indispensable. Experimentally, atomistic dynamics on submillisecond timescales are notoriously difficult to access, making computer simulations the method of choice. Molecular dynamics (MD) simulations of biomolecular systems are nowadays routinely used to study the mechanisms underlying biological function in atomic detail. Examples reach from membrane channels [28], microtubules [20], and whole ribosomes [4] to subcellular organelles [43]. Recently, the first MD simulation of an entire gene was reported, comprising about a billion of atoms [21].

Apart from system size, the scope of such simulations is limited by model accuracy and simulation length. Particularly the accurate treatment of electrostatic interactions is essential to properly describe a biomolecule's functional motions. However, these interactions are numerically challenging for two reasons.

First, their long-range character (the potential drops off slowly with $1/r$ with distance r) renders traditional cut-off schemes prone to artifacts, such that grid-based Ewald summation methods were introduced to provide an accurate solution in 3D periodic boundaries. The current standard is the Particle Mesh Ewald (PME) method that makes use of fast Fourier transforms (FFTs) and scales as $N \cdot \log N$ with the number of charges N [11]. However, when parallelizing PME over many compute nodes, the algorithm's communication requirements become more limiting than the scaling with respect to N. Because of the involved FFTs, parallel PME requires multiple all-to-all communication steps per time step, in which the number of messages sent between p processes scales with p^2 [29]. For the PME algorithm included in the highly efficient, open source MD package GROMACS [42], much effort has been made to reduce as much as possible the all-to-all bottleneck, e.g. by partitioning the parallel computer in long-range and short-range processors, which reduces the number of messages involved in all-to-all communication [17]. Despite these efforts, however, even for multimillion atom MD systems on modern hardware, performance levels off beyond several thousand cores due to the inherent parallelization limitations of PME [30, 42, 45].

The second challenge is the tight and non-local coupling between the electrostatic potential and the location of charges on the protein, in particular titratable/protonatable groups that adapt their total charge and potentially also their charge distribution to their current electrostatic environment. Hence, all protonation states are closely coupled, depend on pH, and therefore the protonation/deprotonation dynamics needs to be taken into account during the simulation. Whereas most MD simulations employ fixed protonation states for each titratable group, several dynamical schemes have been introduced [8, 13, 14, 23, 33, 37] that use a protonation coordinate λ to distinguish the protonated from the deprotonated state. Here, we follow and expand the λ-dynamics approach of Brooks et al. [27] and treat λ as an additional degree

of freedom in the Hamiltonian with mass m_λ. Each protonatable group is associated with its own λ "particle" that adopts continuous values in the interval [0, 1], where the end points around $\lambda = 0$ and $\lambda = 1$ correspond to the physical protonated or deprotonated states. A barrier potential with its maximum at $\lambda \approx 0.5$ serves two purposes. (1) It reduces the time spent in unphysical states, and (2) it allows to tune for optimal sampling of the λ coordinate by adjusting its height [8, 9]. Current λ-dynamics simulations with GROMACS are however limited to small system sizes with a small number n_λ of protonatable groups [7–9], as the existing, PME-based implementation (see www.mpibpc.mpg.de/grubmueller/constpH) needs an extra PME mesh evaluation per λ group and suffers from the PME parallelization problem. While these extra PME evaluations can be overcome for the case where only the charges differ between the states, for the most general case of chemical alterations this is not possible.

Without the PME parallelization limitations, a significantly higher number of compute nodes could be utilized, so that both larger and more realistic biomolecular systems would become accessible. The Fast Multipole Method [15] (FMM) is a method that by construction parallelizes much better than PME. Beyond that, the FMM can compute and communicate the additional multipole expansions that are required for the local charge alternatives of λ groups with far less overhead as compared to the PME case. This makes the communicated volume (extra multipole components) somewhat larger, but no global communication steps are involved as in PME, where the global communication volume grows linearly with n_λ and quadratic with p. We also considered other methods that, like FMM, scale linearly with the number of charges, as e.g. multigrid methods. We decided in favor of FMM, because it showed better energy conservation and higher performance in a comparison study [2].

We will now introduce λ-dynamics methods and related work to motivate the special requirements they have on the electrostatics solver. Then follows an overview of our FMM-based solver and the design decisions reflecting the specific needs of MD simulation. We will describe several of the algorithmical and hardware-exploiting features of the implementation such as error control, automatic performance tuning, the lightweight tasking engine, and the CUDA-based GPU implementation.

2 Chemical Variability and Protonation Dynamics

Classical MD simulations employ a Hamiltonian $\mathcal{H}$ that includes potential terms modeling the bonded interactions between pairs of atoms, the bond angle interactions between bonded atoms, and the van der Waals and Coulomb interactions between all pairs of atoms. For conventional, force field based MD simulations, the chemistry of molecules is fixed during a simulation because chemical changes are not described by established biomolecular force fields. Exceptions are alchemical transformations [36, 38, 46, 47], where the system is either driven from a state A described by Hamiltonian $\mathcal{H}_\mathrm{A}$ to a slightly different state B (with $\mathcal{H}_\mathrm{B}$) via

www.dbooks.org

a λ parameter that increases linearly with time, or where A/B chimeric states are simulated at several fixed λ values between $\lambda = 0$ and $\lambda = 1$, as e.g. in thermodynamic integration [24]. The A → B transition is described by a combined, λ-dependent Hamiltonian

$$\mathcal{H}_{\mathrm{AB}}(\lambda) = (1-\lambda)\mathcal{H}_{\mathrm{A}} + \lambda\mathcal{H}_{\mathrm{B}}. \tag{1}$$

In these simulations, which usually aim at determining the free energy difference between the A and B states, the value of λ is an input parameter.

In contrast, with λ-dynamics [16, 25, 27], the λ parameter is treated as an additional degree of freedom with mass m, whose 1D coordinate λ and velocity $\dot{\lambda}$ evolve dynamically during the simulation. Whereas in a normal MD simulation all protonation states are fixed, with λ-dynamics, the pH value is fixed instead and the protonation state of a titratable group changes back and forth during the simulation in response to its local electrostatic environment [23, 39]. If two states (or *forms*) A and B are involved in the chemical transition, the corresponding Hamiltonian expands to

$$\mathcal{H}(\lambda) = (1-\lambda)\mathcal{H}_{\mathrm{A}} + \lambda\mathcal{H}_{\mathrm{B}} + {}^{m}/_{2}\dot{\lambda}^{2} + V_{\mathrm{bias}}(\lambda) \tag{2}$$

with a bias potential V_{bias} that is calibrated to reflect the (experimentally determined) free energy difference between the A and B states and that optionally controls other properties relating to the A ⇋ B transitions [8]. With the potential energy part V of the Hamiltonian, the force acting on the λ particle is

$$f_{\lambda} = -\frac{\partial V}{\partial \lambda}. \tag{3}$$

If coupled to the protonated and deprotonated form of an amino acid side chain, e.g., λ-dynamics enables dynamic protonation and deprotonation of this side chain in the simulation (see Fig. 1 for an example), accurately reacting to the electrostatic environment of the side chain. More generally, also alchemical transformations beyond protons are possible, as well as transformations involving more than just two forms A and B. Equation 2 shows the Hamiltonian for the simplest case of a single protonatable group with two forms A and B, but we have extended the framework to multiple protonatable groups using one λ_i parameter for each chemical form [7–9].

2.1 Variants of λ-Dynamics and the Bias Potential

The key aim of λ-dynamics methods is to allow for dynamic protonation, but there are three areas in which the implementations differ from each other. These are the coordinate system used for λ, the type of the applied bias potential, and how λ is coupled to the alchemical transition. Before we discuss the different choices, let

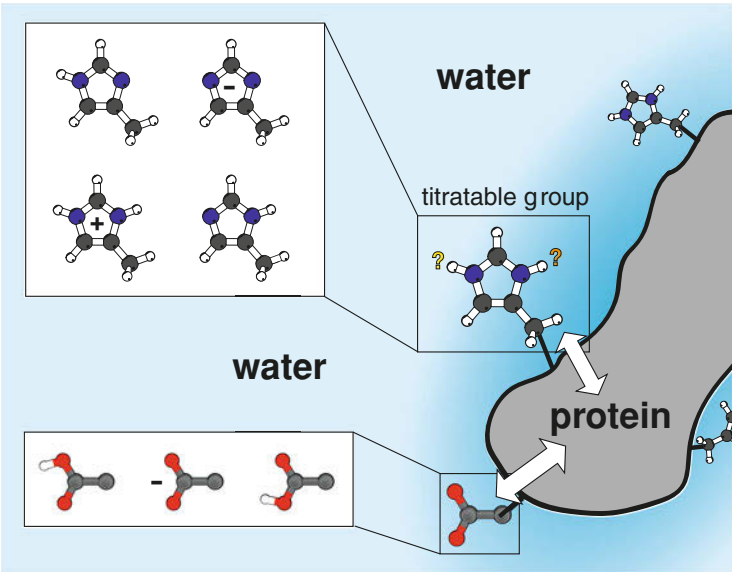


Fig. 1 Simplified sketch of a protein (right, grey) in solution (blue) with several protonatable sites (ball-and-stick representations) of which a histidine (top left) and a carboxyl group (bottom left) are highlighted. The histidine site contains four forms (two neutral, two charged), whereas the carboxyl group contains three forms (two neutral, one negatively charged). In λ-dynamics, the lambdas controls how much of each form is contributing to a site. Atom color coding: carbons-black, hydrogens/protons-white, oxygens-red, nitrogens-blue

us define two terms used in the context of chemical variability and protonation. We use the term **site** for a part of a molecule that can interconvert between two or more chemically different states, e.g. the protonated and deprotonated forms of an aminoacid. Additionally, we call each of the chemically different states of a site a **form**. For instance, a protonatable group is a site with at least two forms A and B, a protonated form A and a deprotonated form B.

2.1.1 The Coordinate System for λ

Based on the coordinate system in which λ lives (or on the dynamical variables used to express λ), we consider three variants of λ-dynamics listed in Table 1. The *linear* variant is conceptually most straightforward, but it definitely needs a bias potential to constrain λ to the interval [0..1]. The circular coordinate system for λ used in the *hypersphere* variant automatically constrains the range of λ values to the desired interval, however one needs to properly correct for the associated circle entropy [8]. The *Nexp* variant implicitly fulfils the constraints on the N_{forms} individual lambdas (Eq. 4) for sites that are allowed to transition between N_{forms}

www.dbooks.org

Table 1 Three variants of λ-dynamics are considered

Variant name	Ref.	Dynamical variable	Geometric picture
Linear	[9]	λ	λ lives on a constricted linear interval, e.g. [0..1]
Hypersphere	[8]	θ	λ lives on a circle
Brooks' Nexp	[26]	ϑ	No simple geometric interpretation

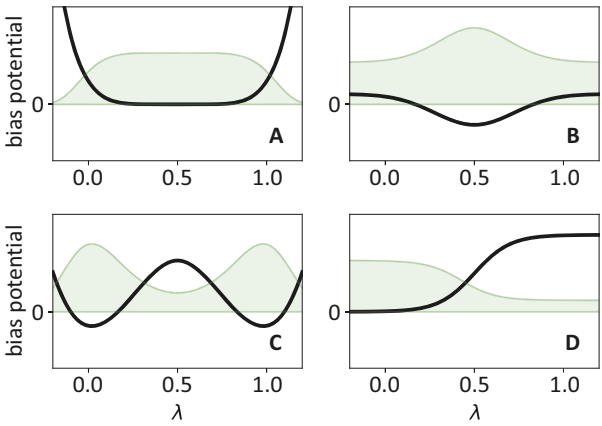


Fig. 2 Qualitative sketches of individual bias potentials (black) that fulfil some of the requirements (1)–(5), and resulting equilibrium distributions of λ values (green)

different forms ($N_{\text{forms}} = 2$ in the case of simple protonation), such that no additional constraint solver for the λ_i is needed.

2.1.2 The Bias Potential

The bias potential $V_{\text{bias}}(\lambda)$ that acts on λ fulfils one or more of the following tasks.

1. If needed, it limits the accessible values of λ to the interval [0..1], whereas slight fluctuations outside that interval may be desirable (Fig. 2a).
2. It cancels out any unwanted barrier at intermediate λ values (b)
3. It takes care that the resulting λ values cluster around 0 or 1, suppressing values between about 0.2 and 0.8 (c)
4. It regulates the depth and width of the minima at 0 and 1, such that the resulting λ distribution fits the experimental free energy difference between protonated and deprotonated form (c + d).
5. It allows to tune for optimal sampling of the λ space by adjusting the barrier height at $\lambda = 0.5$ (c)

Taken together, the various contributions to the barrier potential might look like the example given in Fig. 3 for a particular λ in a simulation.

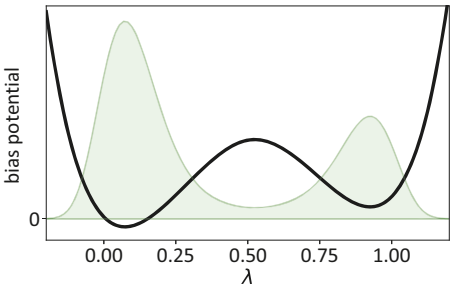


Fig. 3 Qualitative sketch of a bias potential (black) that fulfils all requirements (1)–(5) with resulting equilibrium distribution of λ values (green)

2.1.3 How λ Controls the Transition Between States

The λ parameter can either be coupled to the *transition* itself between two forms (as in [8, 9]), then $\lambda = 0$ corresponds to form A and $\lambda = 1$ to form B. Alternatively, each form gets assigned its own λ_α with $\alpha \in \{A, B\}$ as *weight* parameter. In the latter case one needs extra constraints on the weights similar to

$$\sum \lambda_\alpha = 1, \qquad 0 \leq \lambda_\alpha \leq 1, \tag{4}$$

such that only one of the physical forms A or B is fully present at a time. For the examples mentioned so far, with just two forms, both approaches are equivalent and one would rather choose the first one, because it involves only one λ and needs no extra constraints.

If, however, a site can adopt more than two chemically different forms, the weight approach can become more convenient as it allows to treat sites with any number N_{forms} of forms (using a number of N_{forms} independent λ parameters). Further, it does not require that the number of forms is a power of two ($N_{\text{forms}} = 2^{N_\lambda}$) as in the transition approach.

2.2 *Keeping the System Neutral with Buffer Sites*

In periodic boundary conditions as typically used in MD simulations, the electrostatic energy is only defined for systems with a zero net charge. Therefore, if the charge of the MD system changes due to λ mediated (de)protonation events, system neutrality has to be preserved. With PME, any net charge can be artificially removed by setting the respective Fourier mode's coefficient to zero, so that also in these cases a value for the electrostatic energy can be computed. However, it is merely the energy of a similar system with a neutralizing background charge added. Severe simulation artifacts have been reported as side effects of this approach [19].

As an alternative, a charge buffer can be used that balances the net charge of the simulation system arising from fluctuating charge of the protonatable sites [9, 48]. A reduced number of n_{buffer} buffer sites, each with a fractional charge $|q| \leq 1e$ (e.g.

www.dbooks.org

via $H_2O \leftrightharpoons H_3O^+$), was found to be sufficient to neutralize the N_{sites} protonatable groups of a protein with $n_{\text{buffer}} \ll N_{\text{sites}}$. The total charge of these buffer ions is coupled to the system's net charge with a holonomic constraint [9]. The buffer sites should be placed sufficiently far from each other, such that their direct electrostatic interaction through the shielding solvent is negligible.

3 A Modern FMM Implementation in C++ Tailored to MD Simulation

High performance computing (HPC) biomolecular simulations differ from other scientific applications by their comparatively small particle numbers and by their extremely high iteration rates. With GROMACS, when approaching the scaling limit, the number of particles per CPU core typically lies in the order of a few hundred, whereas the wall-clock time required for computing one time step lies in the range of a millisecond or less [42]. In MD simulations with λ-dynamics, the additional challenge arises to calculate the energy and forces from a Hamiltonian similar to Eq. 2, but for N protonatable sites, in an efficient way. In addition to the Coulomb forces on the regular charged particles, the electrostatic solver has to compute the forces on the N λ particles as well [8] via

$$\begin{aligned} f_{\lambda_i} = -\frac{\partial V_C}{\partial \lambda_i} &= -\frac{\partial V_C(\lambda_1, \ldots, \lambda_{i-1}, \lambda_i, \lambda_{i+1}, \ldots \lambda_N)}{\partial \lambda_i} \\ &= -\Big[V_C(\lambda_1, \ldots, \lambda_{i-1}, \lambda_i = 1, \lambda_{i+1}, \ldots \lambda_N) \\ &\quad - V_C(\lambda_1, \ldots, \lambda_{i-1}, \lambda_i = 0, \lambda_{i+1}, \ldots \lambda_N) \Big] \end{aligned} \tag{5}$$

Accordingly, with λ-dynamics, for each of the λ_i's, the energies of the pure (i.e., $\lambda_i = 0$ and $\lambda_i = 1$) states have to be evaluated while keeping all other lambdas at their actual fractional values.

The aforementioned requirements of biomolecular electrostatics have driven several design decisions in our C++ FMM, which is a completely new C++ reimplementation of the Fortran ScaFaCoS FMM [5]. Although several other FMM implementations exist [1, 50], none of them is prepared to compute the potential terms needed for biomolecular simulations with λ-dynamics.

Although our FMM is tailored for usage with GROMACS, it can be used as an electrostatics solver for other applications as well as it comes as a separate library in a distinct Git repository. On the GROMACS side we provide the necessary modifications such that FMM instead of PME can be chosen at run time. Apart from that, GROMACS calls our FMM library via an interface that can also be used by other codes. The development of this library follows three principles. First, the building blocks (i.e., data structures) used in the FMM support each level

of the hierarchical parallelism available on today's hardware. Second, the library provides different implementations of the involved FMM operators depending on the underlying hardware. Third, the library optionally supports λ-dynamics via an additional interface.

3.1 The FMM in a Nutshell

The FMM approximates and thereby speeds up the computation of the Coulomb potential V_C for a system of N charges:

$$V_C \propto \sum_{i}^{N} \sum_{j<i} \frac{q_i q_j}{|\mathbf{r}_i - \mathbf{r}_j|} \tag{6}$$

For that purpose, the FMM divides the simulation box into eight smaller boxes (depth $d = 1$), which are subsequently subdivided into eight smaller boxes again ($d = 2$) and again ($d = 3, 4, \ldots$). The depth d refers to the number of subdivisions. On the lowermost level, i.e. for the smallest boxes (largest d), all interactions between neighboring boxes are directly calculated (these are called the near-field interactions). Interactions with boxes further away are approximated by a multipole expansion of order p (these are called the far-field interactions). A comprehensive description of the FMM algorithm is beyond the scope of this text, however we will shortly describe the basic workflow and the different operators used in the six FMM stages as these will be referred to in the following sections. For a detailed overview of the FMM, see [22]; for an introduction in our C++ FMM implementation see [12].

3.1.1 FMM Workflow

The FMM algorithm consists of six different stages, five of them required for the farfield (FF) and one for the nearfield (NF) (Fig. 4). After setting up the FMM parameters tree depth (d) and multipole order (p), the following workflow is executed.

1. **P2M**: Expand particles into spherical multipole moments ω_{lm} up to order p on the lowest level for each box in the FMM tree. Multipole moments for particles in the same box can be summed into a multipole expansion representing the whole box.
2. **M2M**: Translate the multipole expansion of each box to its parent box inside the tree. Again, multipole expansions with the same box center can be summed up. The translation up the tree is repeated until the root node is reached.
3. **M2L**: Transform remote multipole moments ω_{lm} into local moments μ_{lm} for each box on every level. Only a limited number of interactions for each box on each level is performed to achieve linear scaling.

www.dbooks.org

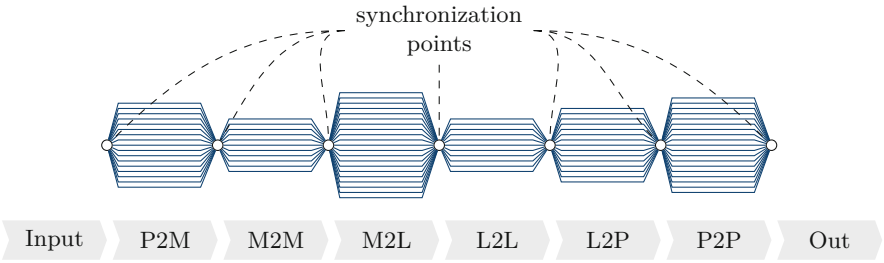


Fig. 4 The classical (sequential) FMM workflow consists of six stages. Only the nearfield (P2P) can be computed completely independent of all other stages. Each farfield stage (P2M, M2M, etc.) depends on the former stage and exhibits different amounts of parallelism. Especially the distribution of multipole and local moments in the tree provide limited parallelism in classical loop-based parallelization schemes

4. **L2L**: Translate local moments μ_{lm} starting from the root node down towards the leaf nodes. Local moments within the same box are summed.
5. **L2P**: After reaching the leaf nodes, the farfield contributions for the potentials Φ_{FF}, forces $\mathbf{F}_{\text{FF}}$, and energy E_{FF} are computed.
6. **P2P**: Interactions between particles within each box and its direct neighbors are computed directly, resulting in the nearfield contributions for the potentials Φ_{NF}, forces $\mathbf{F}_{\text{NF}}$, and energy E_{NF}.

3.1.2 Features of Our FMM Implementation

Our FMM implementation includes special algorithmical features and features that help to optimally exploit the underlying hardware. Algorithmical features are

- Support for open and 1D, 2D and 3D periodic boundary conditions for cubic boxes.
- Support for λ-dynamics (Sect. 2).
- Communication-avoiding algorithms for internode communication via MPI (Fig. 9).
- Automatic tuning of FMM parameters d and p to provide automatic error control and runtime minimization [6] based on a user-provided energy error threshold ΔE (Fig. 10).
- Adjustable tuning to reduce or avoid energy drift (Fig. 11).

Hardware features include

- A performance-portable SIMD layer (Sect. 3.2.1).
- A light-weight, NUMA-aware task scheduler for CPU and GPU tasks (Sect. 3.2.2).
- A GPU implementation based on CUDA (Sect. 3.4).

3.2 Utilizing Hierarchical Parallelism

3.2.1 Intra-Core Parallelism

A large fraction of today's HPC peak performance stems from the increasing width of SIMD vector units. However, even modern compilers cannot generate fully vectorized code unless the data structures and dependencies are very simple. Generic algorithms like FFTs or basic linear algebra can be accelerated by using third-party libraries and tools specifically tuned and optimized for a multitude of different hardware configurations. Unfortunately, the FMM data structures are not trivially vectorizable and require careful design. Therefore, we developed a performance-portable SIMD layer for non-standard data structures and dependencies in C++.

Using only C++11 language features without third-party libraries allows to fine-tune the abstraction layer for the non-trivial data structures and achieve a better utilization. Compile-time loop-unrolling and tunable stacking are used to increase out-of-order execution and instruction-level parallelism. Such optimizations depend heavily on the targeted hardware and must not be part of the algorithmic layer of the code. Therefore, the SIMD layer serves as an abstraction layer that hides such hardware-specifics and that helps to increase code readability and maintainability. The requested SIMD width ($1\times$, $2\times$, ..., $16\times$) and type (float, double) is selected at compile time. The overhead costs and performance results are shown in Fig. 5. The baseline plot (blue) shows the costs of the M2L operation (float) without any vectorization enabled. All other plots show the costs of the M2L operation (float) and 16-fold vectorization (AVX-512). Since the runtime of the M2L operation is limited by the loads of the M2L operator, we try to amortize these costs by utilizing multiple ($2\times \ldots 6\times$) SIMDized multipole coefficient matrices together with a single operator via unrolling (stacking). As can be seen in Fig. 5, unrolling the multipole coefficient matrices $2\times$ (red), we reach the minimal computation time and the expected 16-fold speedup. Additional unroll factors ($3\times \ldots 6\times$) will not improve performance due to register spilling. To reach optimal performance, it is required to reuse (cache) the M2L operator for around 300 (or more) of these steps.

3.2.2 Intra-Node and Inter-Node Parallelism

To overcome scaling bottlenecks of a pragma-based loop-level parallelization (see Fig. 4), our FMM employs a lightweight tasking framework purely based on C++. Being independent of other third-party tasking libraries and compiler extensions allows to utilize resources better, since algorithm-specific behavior and data-flow can be taken into account. Two distinct design features are a type-driven priority scheduler and a static dataflow dispatcher. The scheduler is capable of prioritizing tasks depending on their type at compile time. Hence, it is possible to prioritize vertical operations (like M2M and L2L) in the tree. This reduces the runtime twofold. First, it reduces the scheduling overhead at runtime by avoiding costly

www.dbooks.org

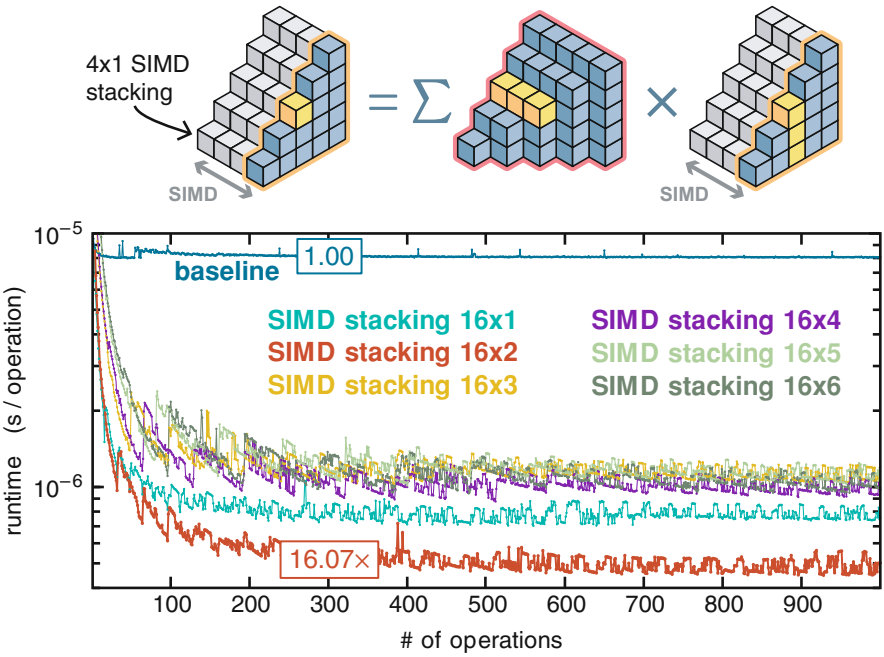


Fig. 5 M2L operation benchmark for vectorized data structures with multipole order $p = 10$ on an Intel Xeon Phi 7250F CPU for a float type with 16× SIMD (AVX-512). The benchmarks shows the performance of different SIMD/unrolling combinations. E.g. the red curve (SIMD stacking 16×2) utilizes 16-fold vectorization together with twofold unrolling For a sufficient number (around 300) of vectorized operations, a 16-fold improvement can be measured for the re-designed data structures

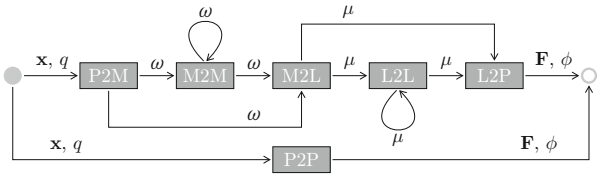


Fig. 6 The data flow of the FMM still consists of six stages. However, synchronization now happens on a fine-grained level and not only after each full stage is completed. This allows to overlap parts that exhibit poor parallelization with parts that show a high degree of parallel code. The dependencies of such a data flow graph can be evaluated and even prioritized at compile time

virtual function calls. Second, since the execution of the critical path is prioritized, the scheduler ensures that a sufficient amount of independent parallelism gets generated. The dataflow dispatcher defines the dependencies between tasks—a data flow graph—also at compile time (see Fig. 6). Together with loadbalancing and workstealing strategies, even a non-trivial FMM data flow can be executed. For compute-bound problems this design shows virtually no overhead. However, in MD

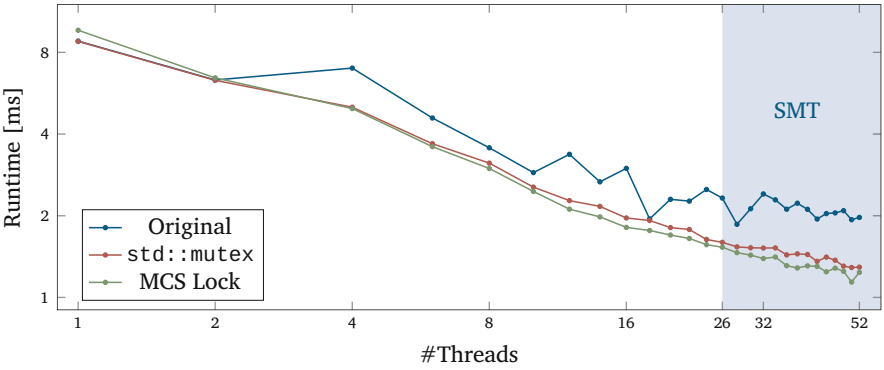


Fig. 7 Intranode FMM benchmark for 1000 particles, multipole order $p = 1$ and tree depth $d = 3$ on a 2x26-core Intel Xeon Platinum 8170 CPU. When using MCS locks, simultaneous multithreading and 50 threads, the overall improvement compared to the original implementation reaches >40%, translating into a reduction in runtime from 1.93 ms down to 1.14 ms

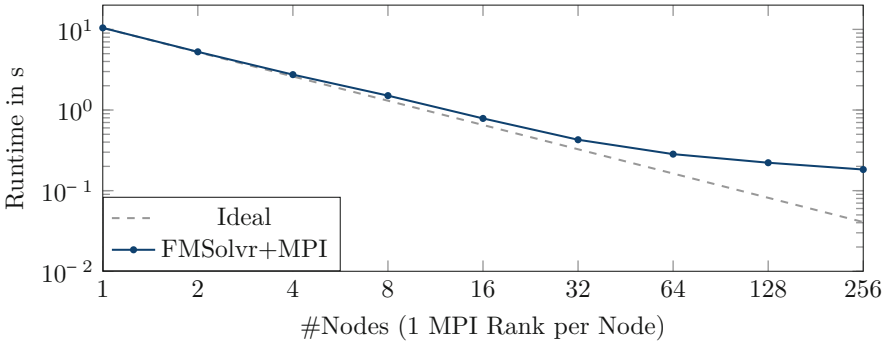


Fig. 8 Initial internode FMM benchmark for 1,000,000 particles, multipole order $p = 3$ and tree depth $d = 5$ with one MPI rank per compute node of the JURECA cluster

we are interested in smaller particle systems with only a few hundred particles per compute node. Hence, we have to take even more hardware constraints into account. Performance penalties due to the memory hierarchy (NUMA) and costs to access memory in a shared fashion via locks introduce additional overhead. Therefore, we extended also our tasking framework with NUMA-aware memory allocations, workstealing and scalable Mellor-Crummey Scott (MCS) locks [35] to enhance the parallel scalability over many threads, as shown in Fig. 7.

In the future, we will extend our tasking framework so that tasks can also be offloaded to local accelerators like GPUs, if available on the node.

For the node-to-node communication via MPI the aforementioned concepts do not work well (see Fig. 8), since loadbalancing or workstealing would create large overheads due to a large amount of small messages. To avoid or reduce

www.dbooks.org

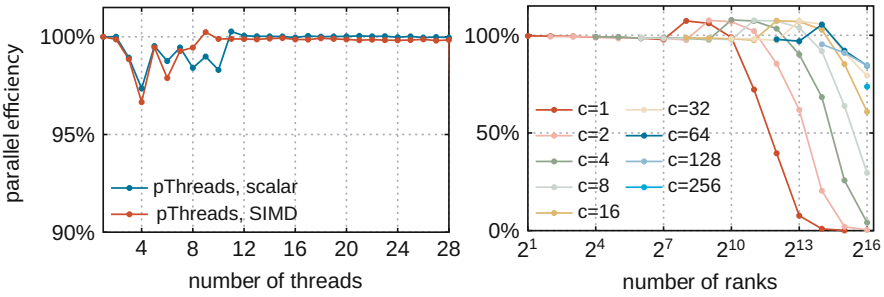


Fig. 9 **Left:** Intranode FMM parallelization—efficiency of different threading implementations. Near field interaction of 114,537 particles in double precision on up to 28 cores on a single node with two 14-core Intel Xeon E5-2695 v3 CPUs. Single precision computation as well as other threading schemes (std::thread, boost::thread, OpenMP) showed similar excellent scaling behavior. The plot has been normalized to the maximum turbo mode frequency which varies with the number of active cores (3.3–2.8 GHz for scalar operation, 3.0–2.6 GHz for SIMD operation). **Right:** Internode parallelization—strong scaling efficiency of a communication avoiding, replication-based workload distribution scheme [10]. Near field interaction of 114,537 particles on up to 65,536 Blue Gene/Q cores using replication factor c. In the initial replication phase, only c nodes within a group communicate. Afterwards, communication is restricted to all pairs of p/c groups. For 65,536 cores, i.e. only 1–2 particles per core initially, a maximum parallel efficiency of 84% (22 ms runtime) is reached for $c = 64$, and the maximal replication factor $c = 256$ yields an efficiency of 73%, while a classical particle distribution ($c = 1$) would require a runtime exceeding 1 min due to communication latency

the latency that comes with each message, we employ a communication-avoiding parallelization scheme [10]. Nodes do not communicate separately with each other, but form groups in order to reduce the total number of messages. At the same time the message size can be increased. Depending on the total number of nodes involved, the group size parameter can be tuned for performance (see Fig. 9).

3.3 Algorithmic Interface

Choosing the optimal FMM parameters in terms of accuracy and performance is difficult if not impossible to do manually as they also depend on the charge distribution itself. A naive choice of tree depth d and multipole order p might either lead to wasting FLOPs or to results that are not accurate enough. Therefore, d and p are automatically tuned depending on the underlying hardware and on a provided energy tolerance ΔE (absolute or relative acceptable error in Coulombic energy). The corresponding parameter set $\{d, p\}$ is computed such that the accuracy is met at minimal computational costs (Fig. 10) [6].

Besides tuning the accuracy to achieve a certain acceptable error in the Coulombic energy for each time step, the FMM can additionally be tuned to reduce the energy drift over time.

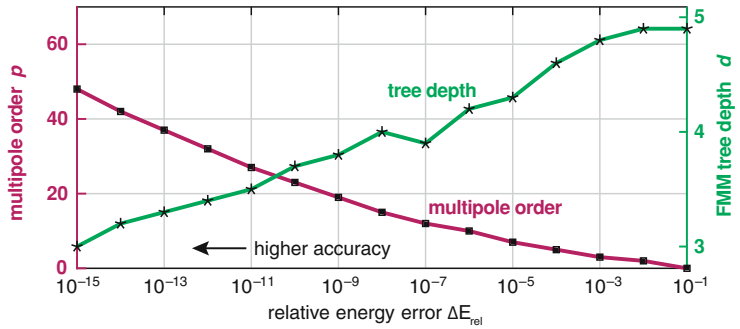


Fig. 10 Depending on a maximum relative or absolute energy tolerance ΔE, the automatic runtime minimization provides the optimal set of FMM input parameters $\{d, p\}$. A lower requested error in energy results in an increased multipole order p (magenta). Since the computational complexity of the farfield operators M2M, M2L and L2L scales with p^3 or even p^4 (depending on the used implementation), the tree depth d is reduced accordingly to achieve a minimal runtime (green). With fractional depths [49], as used here, the runtime can be optimized even more than with integer depths

Whereas multipole orders of about ten yield a comparable drift of the total energy over time as a typical simulation with PME, the drift with FMM can be reduced to much lower levels if desired (Fig. 11).

3.4 CUDA Implementation of the FMM for GPUs

A growing number of HPC clusters incorporate accelerators like GPUs to deliver a large part of the FLOPS. Also GROMACS evolves towards offloading more and more tasks to the GPU, for reasons of both performance and cost-efficiency [31, 32].

For system sizes that are typical for biomolecular simulations, FMM performance critically depends on the M2L and P2P operators. For multipole orders of about eight and larger their execution times dominate the overall FMM runtime (Fig. 12).

Hence, these operators need to be parallelized very efficiently on the GPU. At the same time, all remaining operators need to be implemented on the GPU as well to avoid memory traffic between device (GPU) and host (CPU) that would otherwise become necessary. This traffic would introduce a substantial overhead as a complete MD time step may take just a few milliseconds to execute.

Our encapsulated GPU FMM implementation takes particle positions and charges as input and returns the electrostatic forces on the particles as output. Memory transfers between host and device are performed only at these two points in the calculation step.

www.dbooks.org

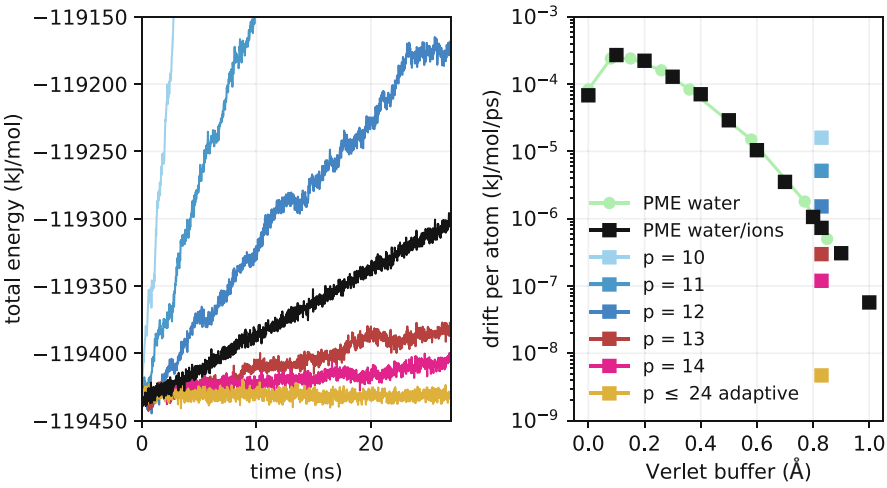


Fig. 11 Observed drift of the total energy for different electrostatics settings. **Left:** evolution of the total energy for PME with order 4, mesh distance 0.113 nm, `ewald-rtol` set to 10^{-5} (black line) as well as for FMM with different multipole orders p at depth $d = 3$ (see legend in the right panel). Test system is a double precision simulation at $T \approx 300$ K in periodic boundaries of 40 Na^+ and 40 Cl^- ions solvated in a 4.07 nm^3 box containing extended simple point charge (SPC/E) water molecules [3], comprising 6740 atoms altogether. Time step $\Delta t = 2$ fs, cutoffs at 0.9 nm, pair-list updated every ten steps. **Right:** Black squares show the drift with PME for different Verlet buffer sizes for the water/ions system using 4×4 cluster pair lists [41]. For comparison, green line shows the same for pure SPC/E water (without ions) taken from Ref. [34]. Influence of different multipole orders p on the drift is shown for a fixed buffer size of 8.3 Å. The GROMACS default Verlet buffer settings yield a drift of $\approx 8 \times 10^{-5}$ kJ/mol/ps per atom for these MD systems, corresponding to the first data point on the left (black square/green circle)

The particle positions and charges are split into different CUDA streams that allow for asynchronous memory transfer to the host. The memory transfer is overlapped with the computation of the spatial affiliation of the octree box.

In contrast to the CPU FMM that utilizes $O(p^3)$ far field operators (M2M, M2L, L2L), the GPU version is based on the $O(p^4)$ operator variant. The $O(p^3)$ operators require less multiplications to calculate the result, but they introduce additional highly irregular data structures to rotate the moments. Since the performance of the GPU FMM at small multipole orders is not limited by the number of floating point operations (Fig. 12) but rather by scattered memory access patterns, we use the $O(p^4)$ operators for the GPU implementation.

We will now outline our CUDA implementation of the operations needed in the various stages of the FMM (Figs. 4, 5, and 6), which starts by building the multipoles on the lowest level with the P2M operator.

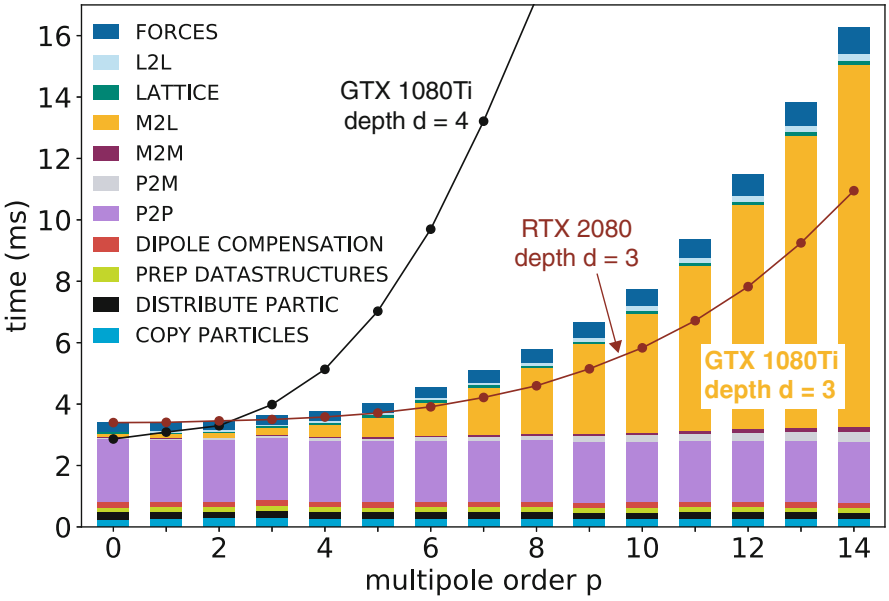


Fig. 12 Colored bars show detailed timings for the various parts of a single FMM step on a GTX 1080Ti GPU for a 103,000 particle system using depth $d = 3$. For comparison, total execution time for $d = 3$ on an RTX 2080 GPU is shown as brown line, whereas black line shows timings for $d = 4$ on a GTX 1080Ti GPU. CUDA parallelization is used in each FMM stage leaving the CPU mostly idle

3.4.1 P2M: Particle to Multipole

The P2M operation is described in detail elsewhere [44]. The large number of registers that is required and the recursive nature of this stage limits the efficient GPU parallelization. The operation is however executed independently for each particle and the requested multipole expansion is gained by summing atomically into common expansion points. The result is precomputed locally using shared memory or intra-warp communication to reduce the global memory traffic when storing the multipole moments. The multipole moments ω, local moments μ and the far field operators $\mathbf{A}$, $\mathbf{M}$, and $\mathbf{C}$ are stored as triangular shaped matrices

$$\omega, \mu, \mathbf{A}, \mathbf{C} \in \mathbb{K}^{p\times p} := \left\{ (x_{lm})_{l=0,\dots,p,\ m=-l,\dots,l} \mid x_{lm} \in \mathbb{C} \right\} \tag{7}$$

and $\mathbf{M} \in \mathbb{K}^{2p\times 2p}$, where p is the multipole order.

To map the triangular matrices efficiently to contiguous memory, their elements are stored as 1D arrays of complex values and the l, m indices are calculated on the fly when accessing the data. For optimal performance, different stages of the FMM require different memory access patterns. Therefore, the data structures are stored redundantly in a Structure of Arrays (SoA) and Array of Structures (AoS) version.

www.dbooks.org

The P2M operator writes to AoS, whereas the far field operators use SoA. A copy kernel, negligible in runtime, does the copying from one structure to another.

3.4.2 M2M: Multipole to Multipole

The M2M operation, which shifts the multipole expansions of the child boxes to their parents, is executed on all boxes within the tree, except for the root node which has no parent box. The complexity of this operation is $O(p^4)$; one M2M operation has the form

$$\omega_{lm}(a') = \sum_{j=0}^{l} \sum_{k=-j}^{j} \omega_{jk}(a) \mathbf{A}_{l-j,m-k}(a - a'), \tag{8}$$

where $\mathbf{A}$ is the M2M operator and a and a' are different expansion center vectors. The operation performs $O(p^2)$ dot products between ω and a part of the operator $\mathbf{A}$. These operations need to be executed in all boxes in the octree, excluding the box on level 0, i.e. the root node. The kernels are executed level wise on each depth, synchronizing between each level. Each computation of the target ω_{lm} for a distinct (l, m) pair is performed in a different CUDA block of the kernel, with threads within a block accessing different boxes sharing the same operator. The operator can be efficiently preloaded into CUDA shared memory and is accessed for different ω_{lm} residing in different octree boxes. Each single reduction step is performed sequentially by each thread. This has the advantage that the partial products are stored locally in registers, reducing the global memory traffic since only $O(p^2)$ elements are written to global memory. It also reduces the atomic accesses, since the results from eight distinct multipoles are written into one common target multipole.

3.4.3 M2L: Multipole to Local

The M2L operator works similarly to M2M, but it requires much more transformations as each source ω is transformed to 189 target μ boxes. The group of boxes to which a particular ω is transformed to is called the interaction set. It contains all child boxes of the direct neighbor boxes of the source's ω parent. The M2L operation is defined as

$$\mu_{lm}(r) = \sum_{j=0}^{p} \sum_{k=-j}^{j} \omega_{jk}(a) \mathbf{M}_{l+j,m+k}(a - r), \tag{9}$$

where r and a are different expansion centers. The operation differs only slightly from M2M in the access pattern but is of the same $O(p^4)$ complexity. As the

M2L runtime is crucial for the overall FMM performance, we have implemented several parallelization schemes. Which scheme is the fastest depends on tree depth and multipole order. The most efficient implementation is based on presorted lists containing interaction box pointers. The lists are presorted so that the symmetry of the operator **M** can be exploited. In **M**, the orthogonal operator elements differ only by their sign. Harnessing this minimizes the number of multiplications and global memory accesses and allows to reduce the number of spawned CUDA blocks from 189 to 54. However, it introduces additional overhead in logic to change signs and computations of additional target μ box positions, so the performance speedup is smaller than 189/54. The kernel is spawned similarly to the M2M kernel performing one dot product per CUDA block preloading the operator **M** into shared memory. The sign changing is done with the help of and additional bitset provided for each operator. Three different parallelization approaches are compared in Fig. 13. Considering the hardware performance bottlenecks of this stage, the limitations highly differ for particular implementations. The naive M2L kernel is clearly bandwidth limited and achieves nearly 500 GB/s for multipole orders larger than ten. This is higher than the theoretical memory throughput of the tested GPU, which is 480 GB/s, due to caching effects. The cache utilization is nearly at 100% achieving 3500 GB/s. However, the performance of this kernel can be enhanced further by moving towards more compute bound regime. With the dynamical approach the performance is mainly limited by the costs of spawning additional kernels. It can be clearly seen with the flat curve shape for multipoles

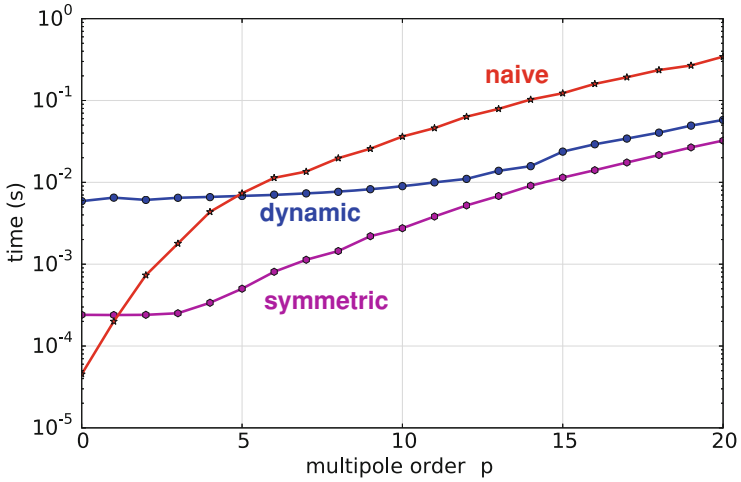


Fig. 13 Comparison of three different parallelization schemes for the M2L operator, which is the most compute intensive part of the FMM algorithm. The naive implementation (red) directly maps the operator loops to CUDA blocks. It beats the other schemes only for orders $p < 2$. Dynamic parallelization (blue) is a CUDA specific approach that dynamically spawns thread groups from the kernels. The symmetric scheme (magenta) represents the FMM tree via presorted interaction lists. It also exploits the symmetry of the M2L operator

www.dbooks.org

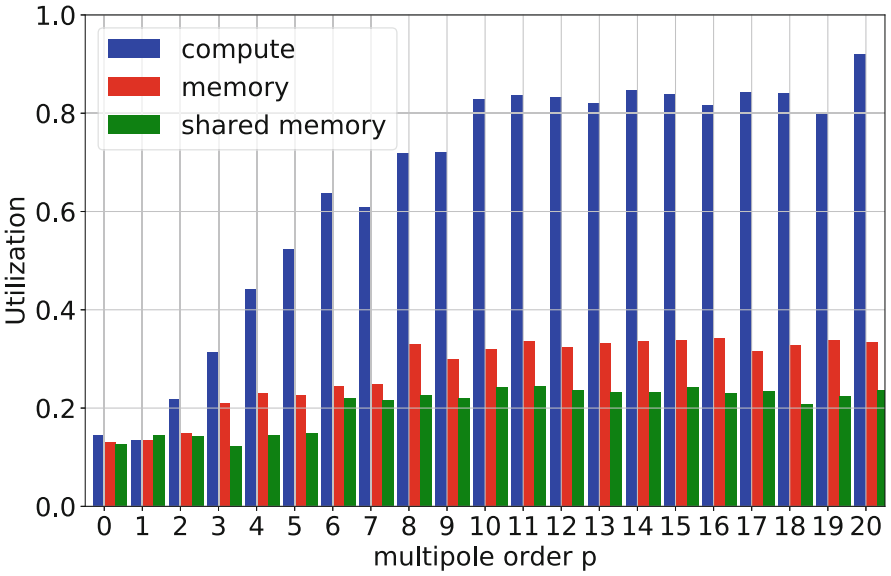


Fig. 14 Hardware utilization of the symmetrical M2L kernel of the GPU-FMM

smaller than 13 in Fig. 13. The hardware utilization for the symmetrical kernel is depicted in Fig. 14. The performance of this kernel depends on the multipole order p, since p^2 is a CUDA gridsize parameter [40]. The values $p < 7$ lead to underutilization of the underlying hardware, however they are mostly not of practical relevance. For larger values the performance is operations bound achieving about 80% of the possible compute utilization.

3.4.4 L2L: Local to Local

The L2L operation is executed for each box in the octree, shifting the local moments from the root of the tree down to the leaves, opposite in direction to M2M. Although the implementation is nearly identical, it achieves slightly better performance than M2M because the number of atomic memory accesses is reduced due to the tree traversing direction. For the L2L operator, the result is written into eight target boxes, whereas M2M gathers information from eight source boxes into one.

3.4.5 L2P: Local to Particles

The calculation of the potentials at particle positions x_i requires evaluating

$$\Phi(x_i) = \sum_{l=0}^{p} \sum_{m=-l}^{l} \mu_{lm} \mathring{\omega}_{lm}^{i}, \quad i = 0, \ldots, N_{\text{box}}\,, \tag{10}$$

where $\mathring{\omega}_{lm}^{i}$ is a chargeless multipole moment of particle at position x_i and N_{box} the number of particles in the box. The complexity of each operation is $O(p^2)$. This stage is similar to P2M since the chargeless moments need to be evaluated for each particle using the same routine for a charge of $q = 1$. The performance is limited by register requirement but like in the P2M stage it runs concurrently for each particle and it is overlapped with the asynchronous memory transfer from device to host.

3.4.6 P2P: Particle to Particle

The FMM computes direct Coulomb interactions only for particles in the leaves of the octree and between particles in boxes that are direct neighbors. These interactions can be computed for each pair of atoms directly by starting one thread for each target particle in the box that sequentially loops over all source particles. An alternative way that better fits the GPU hardware is to compute these interactions for pairs of clusters of size M and N particles, with $M \times N = 32$ the CUDA warp size, as laid out in [41]. The forces acting on the sources and on the targets are calculated simultaneously. The interactions are computed in parallel between all needed box-box pairs in the octree. The resulting speedup of computing all atomic interactions between pairs of clusters instead of using simpler, but longer loops over pairs of atoms is shown in Fig. 15. The P2P kernels are clearly compute bound. The exact performance evaluation of the kernel can be found in [41].

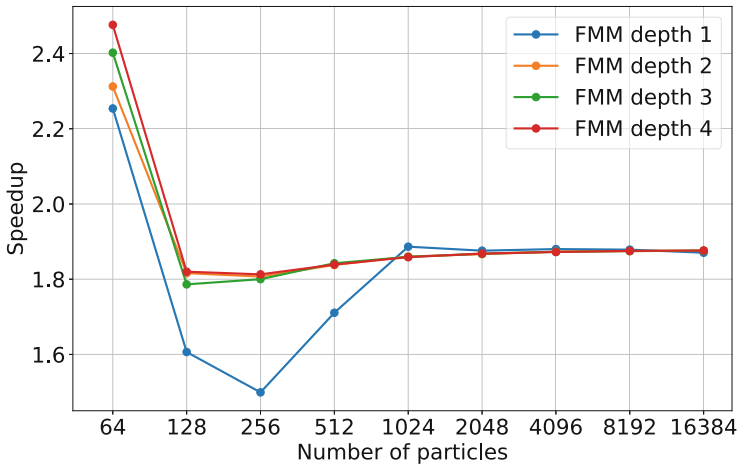


Fig. 15 Speedup of calculating the P2P direct interactions in chunks of $M \times N = 32$ (i.e. for cluster pairs of size M and N) compared to computing them for all atomic pairs (i.e. for "clusters" of size $M = N = 1$). All needed FMM box-box interactions are taken into account

www.dbooks.org

3.5 GPU FMM with λ-Dynamics Support

In addition to the regular Coulomb interactions, with λ-dynamics, extra energy terms for all forms of all λ sites need to be evaluated such that the forces on the λ particles can be derived. The resulting additional operations exhibit a very unstructured pattern that varies depending on the distribution of the particles associated with λ sites. Such a pattern can be described by multiple sparse FMM octrees that additionally interact with each other. The sparsity that emerges from a relatively small size of the λ sites necessitates a different parallelization than for a standard FMM. To support λ-dynamics efficiently, all stages of the algorithm were adapted. Especially, the most compute intense shifting (M2M, L2L) and transformation (M2L) operations need a different parallelization than that of the normal FMM to run efficiently for a sparse octree. Figure 16 shows the runtime of the CUDA parallelized λ-FMM as a function of the system size, whereas Fig. 17 shows the overhead associated with λ-dynamics. The overhead that emerges from addition of λ sites to the simulation system scales linearly with the number of additional sites with a factor of about 10^{-3} per site. This shows that the FMM tree structure fits particularly well the λ-dynamics requirements for flexibility to compute the highly unstructured, additional particle-particle interactions. Note that our λ-FMM kernels still have the potential for more optimizations (at the moment they achieve only about 60% of the efficiency of the regular FMM kernels) such that for the final optimized implementation we expect the costs for the additional sites to be even smaller than what is shown in Fig. 17.

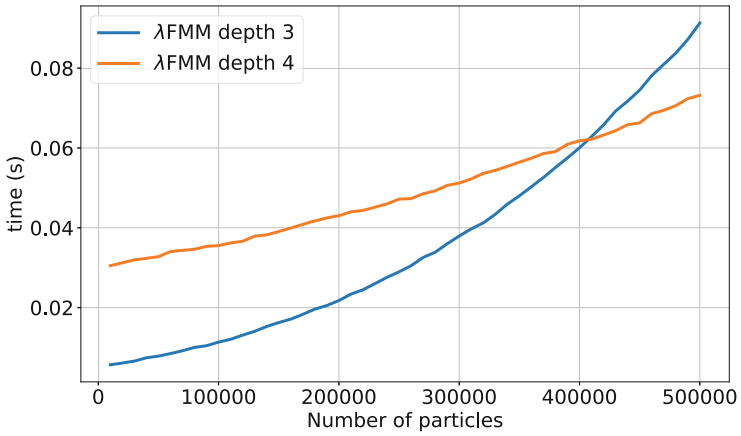


Fig. 16 Absolute runtime of the λ-FMM CUDA implementation. For this example we use one λ site per 4000 particles as estimated from the hen egg white lysozyme model system for constant-pH simulation. Each form of a λ site contains ten particles. The tests were run on a GTX 1080Ti GPU

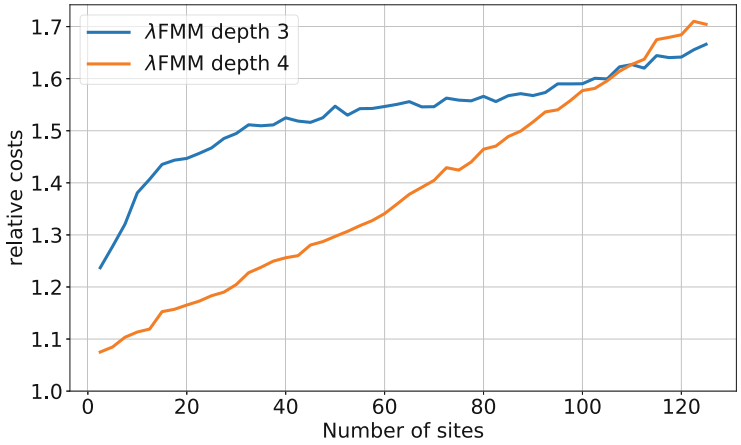


Fig. 17 As Fig. 16, but now showing relative costs of adding λ-dynamics functionality to the regular GPU FMM

4 Conclusions and Outlook

All-atom, explicit solvent biomolecular simulations with λ-dynamics are still limited to comparatively small simulation systems (<100,000 particles) and/or short timescales [7, 9, 18]. To ultimately allow for a realistic (e.g., const-pH) treatment of large biomolecular systems on long timescales, we are developing an efficient FMM that computes the long-range Coulomb interactions, including local charge alternatives for a large number of sites, with just a small overhead compared to the case without λ-dynamics.

Our FMM library is a modern C++11 based implementation tailored towards the specific requirements of biomolecular simulation, which are a comparatively small number of particles per compute core and a very short wall clock time per iteration. The presented implementation offers near-optimal performance on various SIMD architectures, an efficient CUDA version for GPUs, and it makes use of fractional tree depths for optimal performance. In addition to supporting chemical variability via λ-dynamics, it has several more unique features such as a rigorous error control, and based upon that, an automatic performance optimization at runtime. The energy drift resulting from errors in the FMM calculation can be reduced to virtually zero with a newly developed scheme that adapts the multipole expansion order p locally and on the fly in response to the requested maximum energy error. With fixed p, using multipole orders 10–14 yields drifts that are smaller than those observed for typical simulations with PME. We expect the FMM to be useful also for normal MD simulations, as a drop-in PME replacement for extreme scaling scenarios where PME reaches its scaling limit.

The GPU version of our FMM will implicitly use the same parallelization framework as the CPU version. In fact, GPUs will be treated as one of several

www.dbooks.org

resources a node offers (in addition to CPUs), to which tasks can be scheduled. As our GPU implementation is not a monolithic module, it can be used to calculate individual parts of the FMM, like the near-field contribution or the M2L operations of one of the local boxes only, in a fine-grained manner. How much work is offloaded to local GPUs will depend on the node specifications and on how much GPU and CPU processing power is available.

The λ-dynamics module allows to choose between three different variants of λ-dynamics. The dynamics and equilibrium distributions of the lambdas can be flexibly tuned by a barrier potential, whereas buffer sites ensure system neutrality in periodic boundary conditions. Compared to a regular FMM calculation without local charge alternatives, the GPU-FMM with λ-dynamics is only a factor of two slower even for a large (500,000 atom) simulation system with more than 100 protonatable sites.

Although some infrastructure that is needed for out-of-the-box constant-pH simulations in GROMACS still has to be implemented, with the λ-dynamics and FMM modules, the most important building blocks are in place and performing well. The next steps will be to carry out realistic tests with the new λ-dynamics implementation and to thoroughly compare to known results from older studies, before advancing to larger, more complex simulation systems that have become feasible now.

Acknowledgments This work is supported by the German Research Foundation (DFG) Cluster of excellence *Multiscale Imaging* and under the DFG priority programme 1648 *Software for Exascale Computing (SPPEXA)*.

References

1. Agullo, E., Bramas, B., Coulaud, O., Darve, E., Messner, M., Takahashi, T.: Task-based FMM for multicore architectures. SIAM J. Sci. Comput. **36**(1), C66–C93 (2014). https://doi.org/10.1137/130915662
2. Arnold, A., Fahrenberger, F., Holm, C., Lenz, O., Bolten, M., Dachsel, H., Halver, R., Kabadshow, I., Gähler, F., Heber, F., Iseringhausen, J., Hofmann, M., Pippig, M., Potts, D., Sutmann, G.: Comparison of scalable fast methods for long-range interactions. Phys. Rev. E **88**(6), 063308 (2013)
3. Berendsen, H., Grigera, J., Straatsma, T.: The missing term in effective pair potentials. J. Phys. Chem. **91**(24), 6269–6271 (1987)
4. Bock, L.V., Blau, C., Vaiana, A.C., Grubmüller, H.: Dynamic contact network between ribosomal subunits enables rapid large-scale rotation during spontaneous translocation. Nucleic Acids Res. **43**(14), 6747–6760 (2015)
5. Bolten, M., Fahrenberger, F., Halver, R., Heber, F., Hofmann, M., Kabadshow, I., Lenz, O., Pippig, M., Sutmann, G.: ScaFaCoS, C subroutine library. http://scafacos.github.com
6. Dachsel, H.: An error-controlled fast multipole method. J. Chem. Phys. **132**, 119901 (2010). https://doi.org/10.1063/1.3264952
7. Dobrev, P., Donnini, S., Groenhof, G., Grubmüller, H.: Accurate three states model for amino acids with two chemically coupled titrating sites in explicit solvent atomistic constant pH simulations and pKa calculations. J. Chem. Theory Comput. **13**(1), 147–160 (2017). https://doi.org/10.1021/acs.jctc.6b00807

8. Donnini, S., Tegeler, F., Groenhof, G., Grubmüller, H.: Constant pH molecular dynamics in explicit solvent with λ-dynamics. J. Chem. Theory Comput. **7**, 1962–1978 (2011). https://doi.org/10.1021/ct200061r
9. Donnini, S., Ullmann, R.T., Groenhof, G., Grubmüller, H.: Charge-neutral constant pH molecular dynamics simulations using a parsimonious proton buffer. J. Chem. Theory Comput. **12**(3), 1040–1051 (2016). https://doi.org/10.1021/acs.jctc.5b01160
10. Driscoll, M., Georganas, E., Koanantakool, P., Solomonik, E., Yelick, K.: A communication-optimal n-body algorithm for direct interactions. In: Parallel and Distributed Processing Symposium, International, vol. 0, pp. 1075–1084 (2013). https://doi.org/10.1109/IPDPS.2013.108
11. Essmann, U., Perera, L., Berkowitz, M.L., Darden, T., Lee, H., Pedersen, L.G.: A smooth particle mesh Ewald method. J. Chem. Phys. **103**(19), 8577–8593 (1995). https://doi.org/10.1063/1.470117
12. Garcia, A.G., Beckmann, A., Kabadshow, I.: Accelerating an FMM-Based Coulomb Solver with GPUs, pp. 485–504. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-40528-5_22
13. Goh, G.B., Knight, J.L., Brooks, C.L.: Constant pH molecular dynamics simulations of nucleic acids in explicit solvent. J. Chem. Theory Comput. **8**, 36–46 (2012). https://doi.org/10.1021/ct2006314
14. Goh, G.B., Hulbert, B.S., Zhou, H., Brooks III, C.L.: Constant pH molecular dynamics of proteins in explicit solvent with proton tautomerism. Proteins Struct. Funct. Bioinf. **82**(7), 1319–1331 (2014)
15. Greengard, L., Rokhlin, V.: A new version of the fast multipole method for the Laplace equation in three dimensions. Acta Numer. **6**, 229–269 (1997). https://doi.org/10.1017/S0962492900002725
16. Guo, Z., Brooks, C., Kong, X.: Efficient and flexible algorithm for free energy calculations using the λ-dynamics approach. J. Phys. Chem. B **102**(11), 2032–2036 (1998)
17. Hess, B., Kutzner, C., van der Spoel, D., Lindahl, E.: Gromacs 4: algorithms for highly efficient, load-balanced, and scalable molecular simulation. J. Chem. Theory Comput. **4**, 435–447 (2008). https://doi.org/10.1021/ct700301q
18. Huang, Y., Chen, W., Wallace, J.A., Shen, J.: All-atom continuous constant pH molecular dynamics with particle mesh Ewald and titratable water. J. Chem. Theory Comput. **12**(11), 5411–5421 (2016)
19. Hub, J.S., de Groot, B.L., Grubmüller, H., Groenhof, G.: Quantifying artifacts in Ewald simulations of inhomogeneous systems with a net charge. J. Chem. Theory Comput. **10**, 381–390 (2014). https://doi.org/10.1021/ct400626b
20. Igaev, M., Grubmüller, H.: Microtubule assembly governed by tubulin allosteric gain in flexibility and lattice induced fit. eLife **7**, e34353 (2018)
21. Jung, J., Nishima, W., Daniels, M., Bascom, G., Kobayashi, C., Adedoyin, A., Wall, M., Lappala, A., Phillips, D., Fischer, W., Tung, C.S., Schlick, T., Sugita, Y., Sanbonmatsu, K.Y.: Scaling molecular dynamics beyond 100,000 processor cores for large-scale biophysical simulations. J. Comput. Chem. **40**, 1919 (2019)
22. Kabadshow, I., Dachsel, H.: The error-controlled fast multipole method for open and periodic boundary conditions. In: Sutmann, G., Gibbon, P., Lippert, T. (eds.) Fast Methods for Long-Range Interactions in Complex Systems. IAS Series, vol. 6, pp. 85–114. FZ Jülich, Jülich (2011)
23. Khandogin, J., Brooks, C.L.: Constant pH molecular dynamics with proton tautomerism. Biophys. J. **89**(1), 141–157 (2005)
24. Kirkwood, J.G.: Statistical mechanics of fluid mixtures. J. Chem. Phys. **3**(5), 300–313 (1935)
25. Knight, J.L., Brooks III, C.L.: λ-dynamics free energy simulation methods. J. Comput. Chem. **30**(11), 1692–1700 (2009)
26. Knight, J.L., Brooks III, C.L.: Applying efficient implicit nongeometric constraints in alchemical free energy simulations. J. Comput. Chem. **32**(16), 3423–3432 (2011). https://doi.org/10.1002/jcc.21921

www.dbooks.org

27. Kong, X., Brooks III, C.L.: λ-dynamics: a new approach to free energy calculations. J. Chem. Phys. **105**, 2414–2423 (1996). https://doi.org/10.1063/1.472109
28. Kopec, W., Köpfer, D.A., Vickery, O.N., Bondarenko, A.S., Jansen, T.L., de Groot, B.L., Zachariae, U.: Direct knock-on of desolvated ions governs strict ion selectivity in K+ channels. Nat. Chem. **10**(8), 813 (2018)
29. Kutzner, C., van der Spoel, D., Fechner, M., Lindahl, E., Schmitt, U.W., de Groot, B.L., Grubmüller, H.: Speeding up parallel GROMACS on high-latency networks. J. Comput. Chem. **28**(12), 2075–2084 (2007). https://doi.org/10.1002/jcc.20703
30. Kutzner, C., Apostolov, R., Hess, B., Grubmüller, H.: Scaling of the GROMACS 4.6 molecular dynamics code on SuperMUC. In: Bader, M., Bode, A., Bungartz, H.J. (eds.) Parallel Computing: Accelerating Computational Science and Engineering (CSE), pp. 722–730. IOS Press, Amsterdam (2014). https://doi.org/10.3233/978-1-61499-381-0-722
31. Kutzner, C., Páll, S., Fechner, M., Esztermann, A., de Groot, B., Grubmüller, H.: Best bang for your buck: GPU nodes for GROMACS biomolecular simulations. J. Comput. Chem. **36**(26), 1990–2008 (2015). https://doi.org/10.1002/jcc.24030
32. Kutzner, C., Páll, S., Fechner, M., Esztermann, A., de Groot, B.L., Grubmüller, H.: More bang for your buck: improved use of GPU nodes for GROMACS 2018. J. Comput. Chem. **40**(27), 2418–2431 (2019). https://doi.org/10.1002/jcc.26011
33. Lee, M.S., Salsbury Jr, F.R., Brooks III, C.L.: Constant-pH molecular dynamics using continuous titration coordinates. Proteins Struct. Funct. Bioinf. **56**(4), 738–752 (2004)
34. Lindahl, E., Abraham, M., Hess, B., van der Spoel, D.: GROMACS 2019.3 manual. Zenodo (2019). https://doi.org/10.5281/zenodo.3243834
35. Mellor-Crummey, J.M., Scott, M.L.: Algorithms for scalable synchronization on shared-memory multiprocessors. ACM Trans. Comput. Syst. (TOCS) **9**(1), 21–65 (1991)
36. Mermelstein, D.J., Lin, C., Nelson, G., Kretsch, R., McCammon, J.A., Walker, R.C.: Fast and flexible GPU accelerated binding free energy calculations within the AMBER molecular dynamics package. J. Comput. Chem. **39**(19), 1354–1358 (2018)
37. Mertz, J.E., Pettitt, B.M.: Molecular dynamics at a constant pH. Int. J. Supercomputer Appl. High Perform. Comput. **8**(1), 47–53 (1994)
38. Mobley, D.L., Klimovich, P.V.: Perspective: alchemical free energy calculations for drug discovery. J. Chem. Phys. **137**(23), 230901 (2012)
39. Mongan, J., Case, D.A.: Biomolecular simulations at constant pH. Curr. Opin. Struct. Biol. **15**(2), 157–163 (2005)
40. NVIDIA Corporation: NVIDIA CUDA C programming guide (2019). Version 10.1.243
41. Páll, S., Hess, B.: A flexible algorithm for calculating pair interactions on SIMD architectures. Comput. Phys. Commun. **184**, 2641–2650 (2013). https://doi.org/10.1016/j.cpc.2013.06.003
42. Páll, S., Abraham, M.J., Kutzner, C., Hess, B., Lindahl, E.: Tackling exascale software challenges in molecular dynamics simulations with GROMACS. In: Markidis, S., Laure. E. (eds.) Solving Software Challenges for Exascale, pp. 3–27. Springer International Publishing, Cham (2015)
43. Perilla, J.R., Goh, B.C., Cassidy, C.K., Liu, B., Bernardi, R.C., Rudack, T., Yu, H., Wu, Z., Schulten, K.: Molecular dynamics simulations of large macromolecular complexes. Curr. Opin. Struct. Biol. **31**, 64–74 (2015)
44. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes 3rd Edition: The Art of Scientific Computing, 3rd edn. Cambridge University Press, New York (2007)
45. Schulz, R., Lindner, B., Petridis, L., Smith, J.C.: Scaling of multimillion-atom biological molecular dynamics simulation on a petascale supercomputer. J. Chem. Theory Comput. **5**(10), 2798–2808 (2009)
46. Seeliger, D., De Groot, B.L.: Protein thermostability calculations using alchemical free energy simulations. Biophys. J. **98**(10), 2309–2316 (2010)
47. Shirts, M.R., Mobley, D.L., Chodera, J.D.: Alchemical free energy calculations: ready for prime time? Annu. Rep. Comput. Chem. **3**, 41–59 (2007)

48. Wallace, J.A., Shen, J.K.: Charge-leveling and proper treatment of long-range electrostatics in all-atom molecular dynamics at constant pH. J. Chem. Phys. **137**(18), 184105 (2012)
49. White, C.A., Head-Gordon, M.: Fractional tiers in fast multipole method calculations. Chem. Phys. Lett. **257**(5–6), 647–650 (1996). https://doi.org/10.1016/0009-2614(96)00574-X
50. Yokota, R., Barba, L.A.: A tuned and scalable fast multipole method as a preeminent algorithm for exascale systems. CoRR abs/1106.2176 (2011). http://arxiv.org/abs/1106.2176

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



www.dbooks.org

MYX: Runtime Correctness Analysis for Multi-Level Parallel Programming Paradigms



Joachim Protze, Miwako Tsuji, Christian Terboven, Thomas Dufaud, Hitoshi Murai, Serge Petiton, Nahid Emad, Matthias S. Müller, and Taisuke Boku

Abstract In recent years the increasing compute power is mainly provided by rapidly increasing concurrency. Therefore, the HPC community is looking for new parallel programming paradigms to make the best use of current and upcoming machines. Under the Japanese CREST funding program, the post-petascale HPC project developed the XcalableMP programming paradigm, a pragma-based partitioned global address space (PGAS) approach. To better exploit the potential

J. Protze (✉)
RWTH Aachen University, Aachen, Germany
e-mail: protze@itc.rwth-aachen.de

M. Tsuji
RIKEN-CCS, Kobe, Japan
e-mail: miwako.tsuji@riken.jp

C. Terboven
RWTH Aachen University, Aachen, Germany
e-mail: terboven@itc.rwth-aachen.de

T. Dufaud
University of Versailles, Versailles, France
e-mail: thomas.dufaud@uvsq.fr

H. Murai
RIKEN-CCS, Kobe, Japan
e-mail: h-murai@riken.jp

S. Petiton · N. Emad
University of Versailles, Versailles, France
e-mail: serge.petiton@univ-lille.fr; nahid.emad@uvsq.fr

M. S. Müller
RWTH Aachen University, Aachen, Germany
e-mail: mueller@itc.rwth-aachen.de

T. Boku
University of Tsukuba, Tsukuba, Japan
e-mail: taisuke@cs.tsukuba.ac.jp

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_18

concurrency of large scale systems, the mSPMD model was proposed and implemented with the YvetteML workflow description language. When introducing a new parallel programming paradigm, good tool support for debugging and performance analysis is crucial for the productivity and therefore the acceptance in the HPC community. The subject of the MYX project is to investigate which properties of a parallel programming language specification may help tools to highlight correctness and performance issues or help to avoid common issues in parallel programming in the first place. In this paper, we exercise these investigations on the example of XcalableMP and YvetteML.

1 Introduction

Exascale systems are expected to consist of tens of thousands of compute nodes, complemented by specialized accelerators, resulting in system architectures which are heterogeneous on multiple levels. Such architectures challenge the programmer to write multi-level parallel programs, which means employing multiple different paradigms to address each level of parallelism in the system [2]. This ranges from inter-node parallelism in the form of distributed memory parallelism, over shared-memory parallelism to exploit multi-core processors and acceleration units, to vector-style parallelism to target corresponding hardware units. The long-term challenge is to evolve existing and develop new programming models to better support the application development on exascale machines. For different domains and different abstraction levels, various programming models have gained momentum. While there is ongoing research on how to make the currently predominant HPC programming model—namely MPI+X—scale well on such systems, the emerging and more high-level PGAS programming models have shown to deliver high productivity for users and certain types of codes [10]. The JST-CREST funded post-petascale HPC project developed the XcalableMP (XMP) programming paradigm, which combines local and global view PGAS concepts.

The multi-level programming paradigm FP3C [13] as described later in this paper is a solution for post-petascale systems targeting a huge number of processors and the attached acceleration devices. Programmers can express high-level parallelism in the YvetteML (YML) workflow language and employ parallel components written in SPMD programming paradigms like XMP or MPI. Since YML drives and executes multiple SPMD tasks at the same time, this is characterized as mSPMD.

The MYX project aims to combine the know-how and lessons learned of different areas to derive the input necessary to guide the development of future programming models and software engineering methods. Therefore we are developing correctness checking techniques for the XMP programming paradigm and make this analysis also available for the multi-level programming paradigm FP3C.

www.dbooks.org

The contributions of this work are:

- Identify possible correctness issues of XMP applications,
- define an XMP tools interface to provide runtime state and event information to runtime tools,
- extend MUST by XMP specific runtime correctness analyses,
- extend YML to soundly support innovative numeric techniques like UCGLE, and
- provide a workflow to analyze YML+XMP applications driven by the FC2P framework.

The structure of the remaining paper is as follows. In Sect. 2 we introduce the concept of runtime correctness checking and provide an overview of the general implementation of MUST. In Sect. 3 we provide a brief overview of the concepts of the XMP programming paradigm based on an example code. In Sect. 4 we highlight potential correctness issues in XMP applications and what information is needed to analyze those errors. To perform such runtime analysis, a tool like MUST needs state and event information from the XMP runtime system. In Sect. 5 we, therefore, provide a brief overview of the tools interface that we proposed as an extension of XMP to the XMP specification consortium. In Sect. 6 we introduce the concepts of the YML workflow language based on an example code. The unite and conquer method described in Sect. 7 represents an example use case for an mSPMD program implemented with YML for the coarse-grained parallelism and XMP for the implementation of the individual YML tasks. To implement such a method, some extensions of YML are necessary, we also discuss the implications for correctness. In Sect. 8 we present the FP2C framework, which provides a YML+XMP implementation targeted to HPC systems. As MPI is basically the standard for distributed memory HPC systems and those systems also prefer fixed-width jobs, i.e., jobs with a fixed number of processes the FP2C framework is implemented with MPI and dynamically launches MPI processes to fill the requested number of process slots. Finally, in Sect. 9 we present the challenges and solutions to provide runtime correctness analysis in MUST for such a dynamic runtime system.

2 Runtime Correctness Analysis for Parallel Programs

Other than serial programs, parallel programs are affected by non-determinism as an effect of the concurrent execution of multiple threads or processes. For defect programs, this non-determinism can manifest as data races or deadlocks which are not known in serial programming. Different approaches to identify and remove the defects in those programs include static code analysis, model checking, and runtime or post-mortem analysis. Here we want to discuss runtime correctness analysis, where the error detection is performed during the execution of the program.

MUST performs runtime correctness checking for MPI parallel applications. The application developer executes the application under the control of MUST, which

checks at execution time whether the usage of MPI is valid according to the MPI specification. For MPI applications we have shown, that the execution overhead for such runtime analysis is below 20% for the typical use case [8]. Although we aim at complete coverage of the MPI specification, the focus is currently on communication functions.

The overhead of runtime correctness analysis depends significantly on the granularity of the analysis. For MPI, the granularity is quite coarse-grained: there is typically a lot of calculation between MPI function calls which is not analyzed. For data race detection in multithreaded applications, the granularity of analysis is much more fine-grained, as each individual memory access is subject to analysis. Therefore we see a two to hundredfold runtime overhead for data race analysis.

2.1 Runtime Analysis in MUST

For runtime analysis of distributed memory applications, we distinguish three kinds of analyses as shown in Fig. 1. *Local analysis* only needs information from a single application process and can be performed within the application process to avoid unnecessary data transfer. In a multi-threaded application, this analysis potentially needs information from multiple threads. We, therefore, spawn an additional

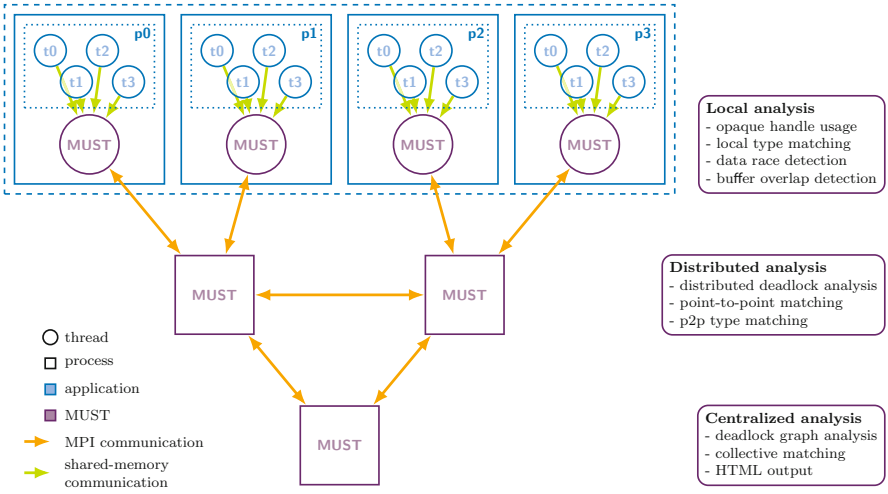


Fig. 1 MUST applied to a hybrid parallel application with four processes $p_0 \ldots p_3$ and four threads $t_0 \ldots t_3$ each. MUST spawns an extra tool thread in each process, which communicates with the additional tool processes using a tree-based overlay network (TBON). Each analysis is performed on the first tool layer that has sufficient information to perform the specific analysis. In the typical setup, communication between MUST processes is performed using MPI. The communication between the threads uses shared memory communication

www.dbooks.org

analysis thread to have all necessary information available. In a single-threaded application or when only the master thread communicates, the local analysis is performed in the application thread and no additional tool thread is spawned. As an example, local type matching compares compile-time information about types used by the application on variable declaration with the runtime information on the corresponding MPI data types used in communication [9].

Distributed analysis needs information from more than one application process. For scalability reasons, the analysis is distributed in the analysis tree and performed in the node where sufficient information is available. As an example, the distributed deadlock detection analysis runs on the parent node of each application process. It works with a distributed state transition system, which is fed with send and receive information for the specific process, as well as completion notification for collective communication [8].

Centralized analysis needs global information from multiple or all application processes. For scalability, a tree reduction analysis is applied where possible, so that the centralized analysis is just the last step in such reduction. An example of such tree reduction is collective matching analysis, where each tree node compares the parameters in collective communication for all child nodes and finally passes one representative to the parent tree node. An example of completely centralized analysis is graph-based deadlock analysis, which we use to visualize the circular dependencies causing a deadlock, but also to verify the presence of a deadlock in the time-out based deadlock detection. This analysis needs information on all pending communication operations but is only executed when a deadlock is detected or suspected.

2.2 Underlying Tool Infrastructure of MUST

MUST intercepts events in the execution of a targeted application to apply the analysis based on the information from these application events. Initially, these events were MPI function calls, but this is now extended to OpenMP and XMP events that are delivered to registered callback functions. Within an application process or thread, the tool can only get active when such an event is delivered. MUST builds on a tree-based overlay network (TBON) communication subsystem, as depicted in Fig. 1. Since the tool cannot make any assumptions, when it will be active on an application process or thread, those nodes communicate only towards the root of the tree. For use cases as point-to-point matching and distributed deadlock detection, the classical TBON communication scheme was extended by horizontal communication within a tool layer. In the current default configuration of MUST, all processes are started together as MPI processes with a common `MPI_COMM_WORLD`. Using the MPI interception layer PMPI, MUST then makes sure, that only the processes intended to execute the application code will continue execution. The tool processes remain in a run loop which waits for incoming events to process. Whenever application code uses `MPI_COMM_WORLD` in an MPI

function call, this communicator is replaced by a sub-communicator representing the application processes. This is transparent for the application, which will never see the additional analysis processes. For performance reasons, the tree layout—including the number of layers and placement of analysis functions— is hard-coded and compiled into a specific instance of the tool. This means that with the current tool infrastructure a dynamic reconfiguration of the tree is not possible at runtime.

3 XcalableMP

XcalableMP (XMP) is a directive-based language extension for Fortran and C languages. Like in OpenMP, parallelism is introduced by the use of directives. If all the directives are ignored by the compiler, a serial program with the same semantics and results should remain. XMP targets parallel programming for distributed memory systems, in contrast to OpenMP targeting shared-memory parallelism. The implementation of XMP in the OmniCompiler is a source to source transformation, which translates the directives into additional code and calls into the XMP runtime library. The XMP runtime library communication is mainly performed using MPI. Therefore, it is in general also possible to use MPI communication in XMP programs or link a library written with XMP into an MPI application.

Listing 1 shows an example of an XMP distributed parallel source code. This example assumes the execution with four processes which are assigned to the nodeset `p`. In XMP, the distribution of a virtual `array` onto nodes is defined as a `template`. An array is then associated with a template using the `align` statement. This defines the distribution of the array over the nodes. Also, the distributed processing is defined by applying the template to a `loop` directive. The iterations of the loop are executed on the different processes according to the distribution assigned to the template `t`.

For parallelization of stencil codes on distributed memory systems, there is typically the need to use a halo as temporary copy for the calculation of the boundary in the local share. XMP supports such behavior with two directives. The `shadow`

Listing 1 Global-view programming: distribute data and work to processes (*nodes*)

```
#pragma xmp nodes p(4)
#pragma xmp template t(0:11)
#pragma xmp distribute t(block) onto p
int B[12];                              // Data Mapping
#pragma xmp align B[i] with t(i)

#pragma xmp loop (i) on t(i)
for(i=0; i<12; i++){                    // Work Mapping
  B[i]=B[i]*2;
}
```

www.dbooks.org

Listing 2 Local-view programming: use coarray notion in C code

```
int a[10]:[*], b[10], c[10];
#pragma xmp nodes p(4)

int me = xmpc_this_image();
int right = (me + 1) % 4, left = (me + 3) % 4;
for(i=0; i<10; i++){
  b[i]=me;                // initialize
}
a[:]:[right] = b[:];      // put to right neighbor
xmp_sync_all(NULL);       // barrier and sync memory
b[:] = a[:];              // local copy
c[:] = a[:]:[left];       // get from left neighbor
```

directive allows to specify the width of the halo for a specific distributed array and the `reflect` directive is to perform the update of the halo.

The functionality described so far is called as global-view programming in XMP. In global-view programming, the application programmer does not need to care where data is located. The array is transparently distributed and accessed. In the suggested workflow, the work is performed where the data is located.

Furthermore, XMP extends Coarray Fortran and makes this functionality also available in C. In XMP this is called local-view programming. To access memory on a different process in local-view programming means to explicitly specify the target process, that holds the image of interest.

The code example in Listing 2 demonstrates how XMP allows using the concept of Coarray in C code. The array `a` is declared as a Coarray of size 10 with an image on each process. The image selector is separated with a colon in the declaration. A classical, local array `b` is initialized in the for loop and then assigned to the Coarray `a`. The slice notation `b[:]` similar to Fortran allows assigning a whole array at once. The assignment to the remote image `right` is semantically a *put* operation. Therefore, the slice notion is not only a convenience feature but allows to perform a single memory transfer in comparison to a for loop, which assigns each array element individually.

4 Correctness Checking for XMP Programs

In this section, we will discuss possible programming errors in XMP applications and how to detect those errors in the code.

4.1 Programming Errors in XMP Programs

For global-view programming we identified a range of possible programming errors that violate restrictions provided for the specific XMP construct. As an example, the `barrier` construct has the following restriction:

- The nodeset specified by the `on` clause must be a subset of the executing nodeset.

The code example in Listing 3 violates this restriction, because the `task` construct limits the executing nodeset to process `p(0)`, while the nodeset specified by the `on` clause is the complete nodeset `p`. The code presents also the MPI idiom with the same semantic. Since only the process with the rank number 0 reaches the barrier, this will finally result in a deadlock for the MPI code.

Besides violations against restrictions imposed by the XMP specification, we also identified possible data races for asynchronous communication. The code example in Listing 4 initializes a distributed array, which is defined with a surrounding halo. The update of the halo is performed asynchronously, because of the `async` clause.

Listing 3 Only a subset of processes participates in a collective barrier operation

```
#pragma xmp task on p(0)
{
  printf("Only executed on rank 0");
  #pragma xmp barrier on p
}
if(rank == 0)
{
  printf("Only executed on rank 0");
  MPI_Barrier(MPI_COMM_WORLD);
}
```

Listing 4 Asynchronously updating the halo can result in a data race

```
int a[16];
#pragma xmp nodes p[4]
#pragma xmp template t[16]
#pragma xmp distribute t onto p
#pragma xmp align a[i] with t[i]
#pragma xmp shadow a[1]

#pragma xmp loop (i) on t[i]
for(int i=0;i<16;i++)
  a[i] = i * 4;

#pragma xmp reflect (a) width (/periodic/1) async(100)
for(int i=0;i<16;i++)
  a[i] = a[i-1] + a[i+1];
#pragma xmp wait(100)
```

www.dbooks.org

Before the asynchronous execution is finished, the stencil access already reads this halo value. It is unclear whether the old or the new value is read, but also whether the old or the new value is sent to the neighbor.

Another possible error arises from the use of the `orthogonal` clause with the `reflect` construct. With this clause, only the orthogonally adjacent halo will be updated, but not the corners or edges of a multidimensional halo. For most stencil applications, this is sufficient and saves a lot of communication, because the corner is located on a different process. If the application nevertheless needs and accesses the value, it will see an uninitialized value there.

For local-view programming, the main risk is data race in the remote memory access. This can occur if different processes access the same memory in the same image without synchronization and one of them modifies the memory. Revisiting the code example in Listing 2, we would have a data race on `a` if we remove the function call to `xmp_sync_all`. The `left` neighbor updates the local image `a` of a process, which would then not be synchronized with the local access to `a` and also not synchronized with the read by the `right` neighbor.

4.2 Correctness Analysis for XMP Programs

Since XMP programs translate to MPI programs in the implementation provided by the omni-compiler, we can apply native MPI correctness analysis to XMP programs. For an application which implements Listing 3, MUST detects a deadlock between an `MPI_Barrier` implementing the XMP barrier directive and an `MPI_Barrier` inserted by the XMP compiler at the end of the task region. Figure 2 shows the deadlock as reported by MUST. The left diagram depicts the cyclic dependency detected by MUST, where `MPI_Barrier` is called with two different communicators. The MUST report provides further details about these communicators, which are created by the XMP implementation. The right diagram provides additional information on the function stack for the two conflicting `MPI_Barrier` calls. `_XMP_Barrier` is the XMP runtime implementation for any explicit or implicit barrier. The graph also shows that this XMP barrier is called from two different locations—lines 15 and 23—in the source code, although the original source code only has 15 lines.

This example emphasizes, that correctness analysis for XMP applications can be done at the MPI level, but is not too useful for the application developer. In other previous work [1, 11] we have seen that we can achieve better results concerning precision and recall if we base the analysis on the semantics of the high-level parallel programming paradigm. Furthermore, the analysis at a higher abstraction level can help to provide more meaningful error reports. In the following, we will see how this applies to XMP.

Analysis in Global-View Programming For the errors, where XMP code might violate restrictions imposed by the XMP specification, we distinguish between static

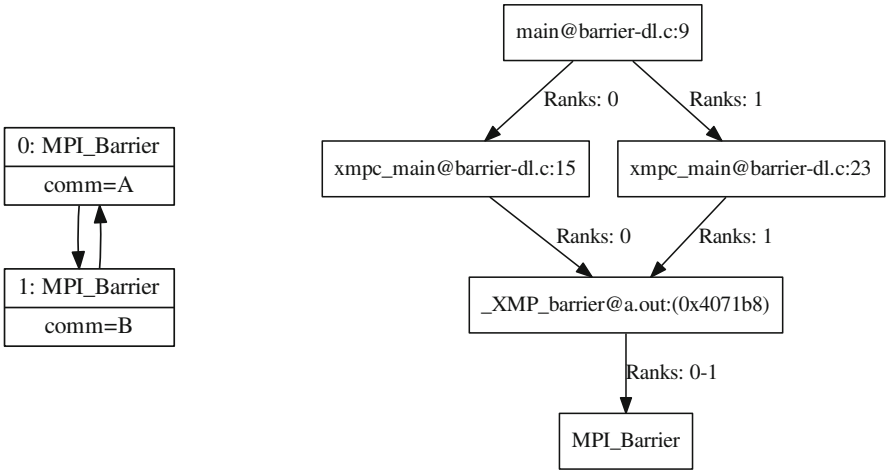


Fig. 2 Deadlock detected by MUST for Listing 3 when only looking at MPI

and dynamic properties of the code. Most of the restrictions on the standalone declarative directives like `nodes`, `template`, or `align` have an impact on static properties of the code. Those restrictions include self-reference in a declaration or lexical name conflicts of handle names with other symbols in a scoping unit. Such restrictions should be enforced by the compiler and result in meaningful compile-time error messages.

Restrictions on dynamic properties depend on the specific value a variable has at runtime. We distinguish between those that involve only the local process and those that involve multiple or all processes. For global-view programming the latter is only the case for collective operations, which require consistent clauses and values among all contributing processes. As for MPI collective communication functions, we analyze this as a reduction analysis, where each node in the TBON compares all incoming events and passes one representative event to the parent node.

All other restrictions on dynamic properties can be analyzed locally. Various XMP constructs have the same restriction as mentioned in Sect. 4.1 for the `barrier` construct. The *nodeset* used for the construct needs to be a subset of the currently executing nodeset. The executing nodeset is the set of nodes executing the current XMP region. The `loop` construct as well as the `task` construct allow to restrict the currently executing nodeset. To perform runtime analysis for such a subset requirement, an analysis tool needs to understand the concept of nodeset and how they can be derived. Listing 5 provides some examples for slicing nodesets. The nodeset `p` consists of the eight processes executing the application, it is also called *entire nodeset*. The nodeset `q` skips the first node in `p` and recruits nodes two to five from `p`. The nodeset `r` is a two-dimensional nodeset, which can be used from two-dimensional domain decomposition. The nodeset `s` is also two-dimensional, but uses only every other node in `p`. Now, checking whether `q[2]` is subset of `s`

www.dbooks.org

Listing 5 Deriving nodesets in XMP

```
#pragma xmp nodes p(8)              // entire nodeset
#pragma xmp nodes q(4)=p(2:5)       // slicing nodes 2-5
#pragma xmp nodes r(2,4)=p(1:8)     // 2-dimensional nodeset
#pragma xmp nodes s(2,2)=p(1:8:2) // skip every other node
```

is a non-trivial question. One method to perform the subset analysis is to expand the given nodeset to the entire nodeset while marking participating nodes. This expansion is not scalable for large numbers of processes, but for some comparison of nodesets we cannot avoid the full expansion.

Analysis in Local-View Programming For local-view programming, our main focus is on data race detection in remote memory access. A data race is commonly understood as concurrent access of multiple execution entities to the same data in memory while at least one access is writing to memory. Concurrent access implies that there is no synchronization between the two memory accesses. We can observe two ways of access to a coarray in XMP, both can be found in the code example in Listing 2. The remote image access denoted by `a[:]:[target]` has write semantics on the target memory for the put operation and read semantics for the get operation. The local image access denoted by `a[:]` or by `a[1]` has the memory access semantics as suggested by the base language. In general, an application might access and modify the local image through a pointer to the local image. Especially, when the array is passed to a library, as a linear algebra library, the access to the local image is out of control of the XMP compiler or runtime system. To detect data races on remote memory access, we need to instrument the local memory accesses as well as tracking all remote memory accesses. Since the conflicting memory access might occur in the library, also the library needs to be instrumented for the runtime data race analysis. This is particularly difficult if the library is only available as a binary.

For data race analysis in MUST, we build on ThreadSanitizer as logging and analysis backend. Memory access instrumentation is performed by clang or GNU compiler during compilation. In addition, we provide high-level synchronization, memory access, and concurrency semantics into the ThreadSanitizer analysis. Therefore we extend the annotation interface used by ThreadSanitizer and Valgrind to feed all necessary information into the analysis. An access to the local image by a remote process should be seen concurrent to any previous access by a different process, that is not synchronized with the current access. Synchronization in XMP is possible with global synchronization, e.g., using the `sync all` directive respectively an `xmp_sync_all()` call, or point to point synchronization, e.g., using the `sync image` directive respectively an `xmp_sync_image()` function call.

5 Tools Interface for XcalableMP

To provide XMP specific runtime information to analysis tools, we designed and implemented the XMP tools interface—XMPT. The interface builds on experiences from the OpenMP tools interface. As an example, the specific environmental variable `XMP_TOOL_LIBRARIES` allows loading an XMP specific tool, when the XMPT interface is available during application execution. This imitates the OMPT specific environmental variable `OMP_TOOL_LIBRARIES` and allows building portable tools, which dynamically adapt to the parallel programming paradigm used by a program.

During startup of an XMP application, the XMP runtime tries to find an XMPT tool, which is identified by the exported function `xmpt_initialize`. Other than in OMPT we don't need a three-way handshake for tool initialization, as the XMP runtime doesn't need to adopt the own initialization in case an XMPT tool is present. Once a tool is found, the XMP runtime calls this function and the tool has the chance to register callbacks for certain XMP events. The current implementation of XMPT provides callbacks for all global-view directives and constructs as well as for coarray memory access and synchronization in local-view programming.

The data mapping identifiers like *node-name*s and *template-name*s are identified by their opaque XMP descriptor handles. To recognize such a descriptor and store information on the descriptor, the XMPT interface allows binding tool data to each XMP descriptor.

The OpenMP specification restricts the OMPT tool to only use OMPT runtime functions, but not to call OpenMP runtime routines like `omp_get_num_threads`, nor to use OpenMP pragmas to implement OMPT callback functions or signal handlers. Without this restriction, the OMPT tool might cause a deadlock in the execution of an OpenMP application, because the OpenMP runtime could hold a lock that it tries to acquire again when the OpenMP runtime function is called. The main difference in this particular aspect is that XMP is initialized explicitly at an early point in the execution by calling `xmp_init`, while OpenMP implementations tend to lazy initialize when the first OpenMP construct or runtime routine is called. For thread-safe initialization, the OpenMP runtime might acquire an initialization lock at any entry to the runtime.

For XMPT there is no restriction on the use of XMP runtime functions so that an XMPT tool can use the variety of inquiry functions to collect all necessary information about the opaque XMP descriptor handles. This allows to query information on XMP specific entities on demand and avoids to transport all available information as arguments to the callbacks. This makes the interface both more compact and more efficient.

www.dbooks.org

6 YvetteML

YvetteML (YML) is a workflow description language for technical or scientific calculation that describes dependencies among tasks.

YML interprets low source code and dependencies between tasks to generate the indicated DAG and execute the task according to the DAG. YML of the original casing is, P2P tasks that are written sequentially in the language it was assumed to run in an environment or a small cluster, of tasks written in a parallel language. By using YML it became possible to run the application on a large scale system. We also developed middleware for porting. The middleware used to implement the mSPMD programming model is OmniRPC-MPI [13]. It provides Remote Procedure Call (RPC) based on MPI and is an extension of the library OmniRPC. Our OmniRPC-MPI middleware is a workflow scheduler to control remote programs which are created for task execution by use of `MPI_Comm_spawn` on request. Control and data flow is implemented using MPI functions such as and `MPI_Send` and `MPI_Recv`.

Listing 6 shows a simple example for a YML program. It invokes a function `add` which takes two double arguments and on return provides the sum in the first argument. The execution starts sequentially, at first $result = 1 + 2$ is calculated. Then execution continues parallel with three concurrent code blocks, separated by `//`. The first code block is just to satisfy the dependency on `ping[0]`, the other two concurrent code blocks execute five parallel iterations each. We can interpret each of the iterations as a task, the `wait` and `notify` statements express dependencies. The YML interpreter generates a DAG as depicted in Fig. 3, where each parallel block and each parallel loop iteration becomes a task. Each of the leaf tasks executes

Listing 6 YvetteML example

```
compute add(result, 1.0, 2.0);   # result <- 1 + 2
par
    notify(ping[0]);
//
    par(i:=0;4)
    do
        wait(ping[i]);
        compute add(result, result, result);
        notify(pong[i]);
    enddo
//
    par(i:=0;4)
    do
        wait(pong[i]);
        compute add(result, result, result);
        notify(ping[i+1]);
    enddo
endpar
```

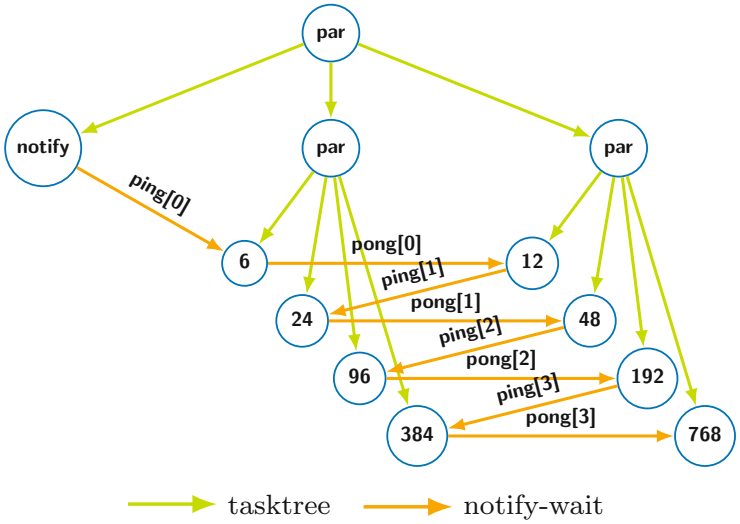


Fig. 3 Graph of tasks as defined by the YvetteML code in Listing 6. The task nodes representing the inner task executing the `compute add` are represented by the result of their computation

one of the `add` functions and the vertices represent the dependencies expressed by the `notify` and `wait` statements. Due to the alternating dependencies between the tasks, this program at the end executes sequentially.

7 Unite and Conquer Approach Using YvetteML

The Unite and Conquer approach was introduced by Emad et al. [6]. The principle of this approach is to make the collaboration of several iterative methods to accelerate the convergence of one of them. This approach can be seen as a model for the design of numerical methods by combining different computational components to work for the same objective, with asynchronous communication among them. Unite implies the combination of different computational components, and conquer represents different components work together to solve one problem. Different independent components with asynchronous communications can be deployed on various platforms such as P2P, cloud and supercomputer systems. The idea of mixing asynchronously restarted Krylov methods using distributed and parallel computing was initially introduced by Guy Edjlali and Serge Petiton [4, 5]. They experimented those hybrid Krylov methods asynchronously on networks of heterogeneous parallel computers (e.g., using two Connection Machines, a CM5 and a CM200 and a network of workstations).

www.dbooks.org

Dividing iterative methods into components coupled with asynchronous communication, as suggested in the Unite and Conquer approach, introduces both numerical and parallel benefits for the components.

Numerical benefits: for conventional deflation and polynomial preconditioned methods, the information used is obtained from previous Arnoldi reduction, and it might be difficult to explore larger subspace. Therefore, the convergence might be slowed down. For the methods implemented with the proposed paradigm, the solving and preconditioning parts are independent. This information applied to the deflation or polynomial preconditioned Solver Components can be different from their own Arnoldi reduction, which improves the flexibility of the algorithms, e.g., much more eigenvalues and larger searching space for the deflation. Hence the limitation of spectral information caused by restarting might be broken down, and faster convergence might be obtained. The numerical benefits for linear and eigensolver are already respectively discussed in [7, 14].

Parallel benefits: parallel performance of iterative methods can be improved by the asynchronous promotion and reduction of synchronizations and global communications, especially the synchronization points for the preconditioning. Separating components improves also the fault tolerance and reusability of algorithms.

7.1 UCGLE

UCGLE (Unite and Conquer GMRES/GMRES-LS method) is a linear equation solver implementation based on the Unite and Conquer approach. It composes mainly three computing components: ERAM, GMRES (Generalized Minimal Residual method), and LS (Least-Squares polynomial method). The GMRES component is used to solve the systems, the LS and ERAM components work as the preconditioning part. The asynchronous communication of this hybrid method among three components reduces the number of overall synchronization points and minimizes global communication. The work-flow of UCGLE with three computing components: The ERAM component computes the desired number of dominant eigenvalues, and then sends them to LS component; the LS component uses these received eigenvalues to generate a new residual vector, and sends it to the GMRES component; the GMRES component uses this residual as a new restarted initial vector for solving the non-Hermitian linear systems. Figure 4 shows the better convergence acceleration of UCGLE compared with preconditioned GMRES. The convergence of UCGLE is accelerated by the LS polynomial preconditioning.

For the use-case of multiple right-hand sides, Wu and Petiton extend this method to m-UCGLE [14]. The m-UCGLE approach furthermore splits the problem into blocks, which are solved individually while feeding their results asynchronously into the computation of the other blocks. This loosely synchronized blocking approach is supported by the general asynchronous feedback loop in the UCGLE approach. Overall this method shows better scalability than other approaches while still profiting from the improved convergence behavior of the UCGLE method.

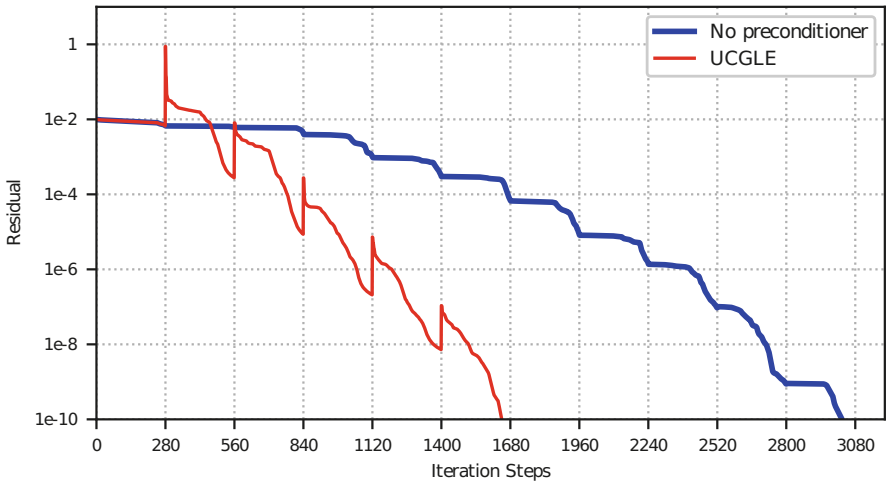


Fig. 4 Convergence comparison of UCGLE method vs. classic GMRES

7.2 Extending YvetteML to Support m-UCGLE

With the current version of YML, the implementation of an m-UCGLE method is not possible due to two limitations: There is no mean for asynchronous communication in YML as needed for the asynchronous feedback loop. YML also provides no way to break early from a YML loop. The latter would be needed to stop iteration at a convergence condition. To make the control flow depending on asynchronous communication, it is necessary to break at multiple levels. Therefore, we propose different kinds of exiting a parallel branch in YML:

1. the application may exit the parallel branch if all the running tasks are completed, e.g., if there are several BGMRES components in parallel to solve linear systems, this parallel section should be exit if all the BGMRES component achieve the convergence;
2. the application may exit the parallel branch if only one task among all is completed, e.g., in the MERAM algorithm, several ERAM components are executed in parallel to approximate the eigenvalues of a matrix, if one of these components approximates enough eigenvalues, the whole parallel section should be exited;
3. the application may exit the parallel branch if only several tasks among all are completed;
4. for the application with multi-level parallelism, we may decide to exit several levels of parallel branches; and
5. the application may exit with saving selected data into the local filesystems, which will improve its fault tolerance and reusability, e.g., lsparams generated by

www.dbooks.org

the B-LSP Component could be saved into local, which will be used for solving the linear systems in future.

By use of the different ways to exit a parallel branch, a unite and conquer algorithm could be implemented with YML. The latter point would even introduce resilience to the YML implementation allowing efficient checkpoint and restart to be defined in the YML description.

8 FP2C

FP2C (Framework for Post-Petascale Computing) is a development and execution environment which supports multi-program methodologies across multiple architectural levels as suggested by Dufaud et al. [3]. FP2C integrates XMP to describe tasks into the workflow environment YML. Therefore FP2C is an implementation of YML to be executed on classical HPC clusters. FP2C is composed of three layers:

1. workflow programming,
2. parallel and distributed programming, and
3. shared-memory parallel programming/accelerator.

The tasks are expected to be executed on sub-clusters or groups of nodes which are tightly connected. These tasks would be hybrid programs with distributed and shared programming models. The workflow scheduler among the sub-clusters or groups invokes and manages the tasks.

The YML backend implementation used for this configuration is OmniRPC-MPI to allow dynamic creation and control of MPI processes needed to executed the YML tasks on an HPC cluster. OmniRPC-MPI is an extension of OmniRPC [12], which supports remote procedure call (RPC) in a grid environment. When the OmniRPC-MPI receives requests to invoke remote programs or to execute tasks on the remote programs, then it handles the requests by calling MPI function such as `MPI_Comm_spawn` to create new processes for the task or `MPI_Send` to notify existing, available processes about the new task.

Figure 5 depicts the execution of a workflow with FP2C. Initially, mpirun only starts the process for the YML scheduler. The scheduler loads the task graph and starts executing the YML program by creating and scheduling YML tasks. Using `MPI_Comm_spawn`, the scheduler creates remote programs with the required number of processes to execute a specific task. To avoid the overhead of process startup and shutdown, the scheduler can reuse an existing group of processes to schedule another task, when the previous task is finished like depicted for task2 and task3. By the use of MPI point-to-point communication, the remote program is informed about the next task to execute and also communicates back about the completion of a task. If some YML tasks need a different number of parallel processes than the previously finished task, FP2C will terminate the remote program to spawn new remote programs as depicted for task1, which is replaced by smaller remote programs to execute task5 and task4.

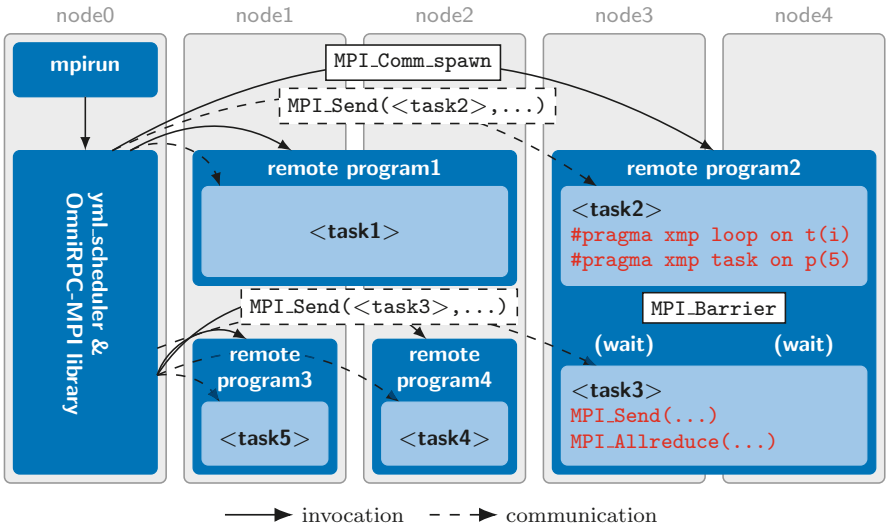


Fig. 5 Execution of an mSPMD application described in the YML control and data flow language in the FP2C implementation

9 Correctness Checking for YML

Correctness checking touches YML at multiple levels:

- The language Yvette, that expresses the graph semantics within YML, is quite similar to the hardware description language Esterel and suffers from similar correctness issues. In this section, we will especially cover potential deadlocks and data races as well as some semantic issues that come with this language.
- The runtime system implementation of YML could also be subject of correctness analysis. The challenge is then to distinguish the behavior of the YML runtime system from application behavior to minimize the analysis overhead.
- Finally, since YML expresses a workflow and runs various modules, it can be of interest to analyze the individual modules separately for correctness.

9.1 Programming Errors in YML Description

With the current specification of YML, the graph defined by a YML graph description can be statically built and therefore also statically analyzed. We identified various possible error patterns in graph descriptions. Possible errors include the use of undefined variables, type miss-match for a variable, but also deadlock due to wait conditions which never receive a signal. Due to the static and self-contained nature of the graph description language, even the possible deadlocks can be identified

www.dbooks.org

statically with data flow analysis. The analysis for those programming errors should be integrated into the YML compiler.

9.2 Challenges of Analyzing the YML Runtime System

Analyzing the runtime system of YML has two mayor challenges for a runtime analysis tool like MUST, which is developed to support the analysis of common HPC applications. The first challenge is to understand the difference between code that represents the YML runtime system and code that belongs to the application code, in this context the YML task code. The bigger challenge is the dynamic MPI characteristic of the YML runtime system, which dynamically creates processes using `MPI_Comm_spawn`, that are then integrated into the execution and should also be supervised by the analysis tool. The analysis tool would also need to understand the resulting new MPI communicators as well as the communication patterns with those spawned processes.

Supporting an application that exposes such dynamic behavior is currently not supported by MUST and the underlying TBON communication layer. The tool would dynamically need to decide about additionally needed analysis processes to extend the TBON. Creating a TBON infrastructure which supports such dynamic application behavior might be subject of a future project.

9.3 Correctness Checking Integrated into FP2C

Since each YML task, invoked by the YML runtime, can be a complete parallel program, such task can have any issue which can also be found in parallel programs. Therefore a developer might want to analyze individual tasks for parallel correctness to identify issues like deadlocks or data race within a task. We introduce a new option for the definition of compute functions into the YML description, which allows applying an analysis tool like MUST to specific YML tasks.

For those selected tasks, the YML scheduler needs to launch additional processes to execute the distributed and centralized analysis of MUST as depicted for remote program2 in Fig. 6. In this specific example, MUST executes both kinds of analysis in a single process. Before launching the FP2C application, the MUST infrastructure needs to be prepared for the execution with each task configuration, which would be done by the `mustrun` execution wrapper for a normal MPI or XMP application. For the execution of a YML task with applied MUST analysis, the remote program controlled by FP2C then needs to select the appropriate prepared configuration of MUST, which is typically done by exporting some environmental variables.

Since we specifically want to analyze the YML task, but not the YML infrastructure, the MPI functions called to implement the FP2C command and control workflow should be ignored by the analysis tool. Some of those MPI functions are

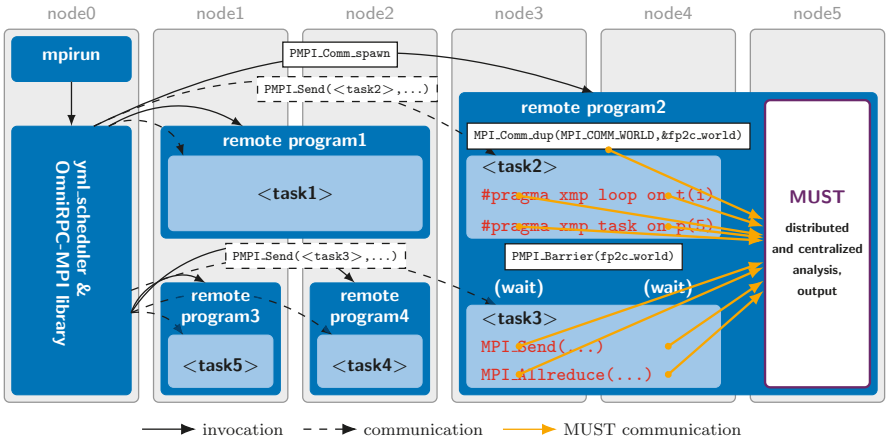


Fig. 6 Integration of MUST into FP2C: the YML scheduler launches selected YML tasks under the control of MUST runtime correctness checking. MUST analyses are applied only to parallel programming constructs used in the task code, but not to MPI communication utilized by the FP2C command and control workflow

used to communicate with the YML scheduler, which is outside of the process controlled by MUST. Such communication to the outside would confuse some analysis performed by MUST. We can avoid analysis of such functions by directly calling into the PMPI interface for functions that implement FP2C functionality. Circumventing MUST analysis for all FP2C-owned MPI communication can result in a deadlock: As Fig. 5 shows, FP2C will execute a barrier at the end of task execution to ensure that all processes finished the execution of the task. For native FP2C execution, it is valid to use `MPI_COMM_WORLD` for this barrier. The MUST analysis process does not know about the barrier and the execution will therefore stall. With FP2C using PMPI calls, MUST will not be able to replace `MPI_COMM_WORLD` by a communicator representing the application processes as it was described in Sect. 2.2. The application processes cannot pass the barrier and the MUST process waits for new messages from the application processes. We could fix this issue by deriving `fp2c_world` from `MPI_COMM_WORLD` using `MPI_Comm_dup` as shown in Fig. 6. The `fp2c_world` communicator can then safely be used by FP2C in PMPI communication calls which are limited to the application processes.

Another challenge when applying MUST to YML tasks is to deal with the output files of MUST. By default, MUST assumes that it is applied to a single MPI application and will write an output file to the current working directory. With FP2C we apply MUST to various YML tasks. To enable the application developer to associate the error report to a specific YML task, we should write the MUST output to a different file per task. The MPI specification defines `int MPI_Pcontrol(const int level, ...)` to allow flexible interaction between MPI application and PMPI tool. It is the responsibility

www.dbooks.org

of the tool to interpret and define the arguments passed to this variadic function. For our use case, we defined usage with `level=8192` and signature `int MPI_Pcontrol(const int level, const char* filename)` to indicate that the analysis results of subsequent application events should be written to the new file name. We also want to make sure that distributed analysis for application events before the pcontrol call write the report into the old file. Therefore we require this pcontrol function call to be collective on the whole application. Currently, we do not require to finish all MPI communication at this point. In the future, we might add some additional pcontrol commands to express certain runtime assertions. Such assertions might include that no outstanding messages are expected or all MPI handles should be released at a certain point.

10 Conclusion

In this paper, we discussed how we can apply runtime correctness checking to emerging multi-level parallel programming languages which try to encounter the challenges of multi-level concurrency of exascale systems. Specifically, we looked into possible correctness issues in XMP applications, which represent the field of PGAS languages. We described how we integrated runtime correctness analysis for XMP applications into the runtime correctness checking tool MUST and therefore specified the new tools interface XMPT for XcalableMP. The workflow description language YML allows to introduce another level of high-level concurrency and therefore better exploit the massive available concurrency of exascale systems. As an example application for such a high-level concurrency workflow, we introduced the unite and conquer method UCGLE. This method improves the convergence behavior of certain solvers of linear equation systems by asynchronously exchanging intermediate results of preconditioner and solver. We introduced FP2C as an implementation of YML targeting HPC systems. We showed how we could integrate MUST runtime analysis to be applied to certain tasks scheduled by the FP2C runtime system and discussed solutions for challenges on the way to a successful workflow.

Acknowledgments The research reported here received funding by the German Research Foundation (DFG) through the priority program 1648 SPPEXA, by the Agence nationale de la recherche (ANR) and by the Japan Science and Technology Agency (JST). The authors would like to thank them for making this research possible.

References

1. Atzeni, S., Gopalakrishnan, G., Rakamaric, Z., Ahn, D.H., Laguna, I., Schulz, M., Lee, G.L., Protze, J., Müller, M.S.: ARCHER: Effectively spotting data races in large openmp applications. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, May 23–27, 2016, pp. 53–62 (2016)
2. Dongarra, J., Beckman, P., Moore, T., Aerts, P., Aloisio, G., Andre, J.C., Barkai, D., Berthou, J.Y., Boku, T., Braunschweig, B., Cappello, F., Chapman, B., Chi, X., Choudhary, A., Dosanjh, S., Dunning, T., Fiore, S., Geist, A., Gropp, B., Harrison, R., Hereld, M., Heroux, M., Hoisie, A., Hotta, K., Jin, Z., Ishikawa, Y., Johnson, F., Kale, S., Kenway, R., Keyes, D., Kramer, B., Labarta, J., Lichnewsky, A., Lippert, T., Lucas, B., Maccabe, B., Matsuoka, S., Messina, P., Michielse, P., Mohr, B., Mueller, M.S., Nagel, W.E., Nakashima, H., Papka, M.E., Reed, D., Sato, M., Seidel, E., Shalf, J., Skinner, D., Snir, M., Sterling, T., Stevens, R., Streitz, F., Sugar, B., Sumimoto, S., Tang, W., Taylor, J., Thakur, R., Trefethen, A., Valero, M., Van Der Steen, A., Vetter, J., Williams, P., Wisniewski, R., Yelick, K.: The international exascale software project roadmap. Int. J. High Perform. Comput. Appl. **25**(1), 3–60 (2011). https://doi.org/10.1177/1094342010391989
3. Dufaud, T., Tsuji, M., Sato, M.: Design of data management for multi SPMD workflow programming model. In: Proceedings of the 4th International Workshop on Extreme Scale Programming Models and Middleware, ESPM2@SC 2018, Dallas, November 11–16, 2018, pp. 9–18 (2018)
4. Edjlali, G., Emad, N., Petiton, S.: Hybrid methods on network of heterogeneous parallel computers. In: Proceedings of the 14th IMACS World Congress, Atlanta (1994)
5. Edjlali, G., Petiton, S., Emad, N.: Interleaved parallel hybrid Arnoldi method for a parallel machine and a network of workstations. In: Conference on Information, Systems, Analysis and Synthesis (ISAS'96), pp. 22–26 (1996)
6. Emad, N., Petiton, S.G.: Unite and conquer approach for high scale numerical computing. J. Comput. Sci. **14**, 5–14 (2016). https://doi.org/10.1016/j.jocs.2016.01.007. https://doi.org/10.1016/j.jocs.2016.01.007
7. Emad, N., Petiton, S., Edjlali, G.: Multiple explicitly restarted arnoldi method for solving large eigenproblems. SIAM J. Sci. Comput. **27**(1), 253–277 (2005)
8. Hilbrich, T., de Supinski, B.R., Nagel, W.E., Protze, J., Baier, C., Müller, M.S.: Distributed wait state tracking for runtime MPI deadlock detection. In: International Conference for High Performance Computing, Networking, Storage and Analysis, SC'13, Denver, November 17–21, 2013, pp. 16:1–16:12 (2013). https://doi.org/10.1145/2503210.2503237. https://doi.org/10.1145/2503210.2503237
9. Hück, A., Lehr, J., Kreutzer, S., Protze, J., Terboven, C., Bischof, C.H., Müller, M.S.: Compiler-aided type tracking for correctness checking of MPI applications. In: 2nd IEEE/ACM International Workshop on Software Correctness for HPC Applications, CORRECTNESS@SC 2018, Dallas, November 12, 2018, pp. 51–58 (2018). https://doi.org/10.1109/Correctness.2018.00011. https://doi.org/10.1109/Correctness.2018.00011
10. Nakao, M., Lee, J., Boku, T., Sato, M.: Productivity and performance of global-view programming with xcalablemp pgas language. In: Proceedings of 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid, pp. 402–409 (2012). https://doi.org/10.1109/CCGrid.2012.118
11. Protze, J., Schulz, M., Ahn, D.H., Müller, M.S.: Thread-local concurrency: a technique to handle data race detection at programming model abstraction. In: Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2018, Tempe, June 11–15, 2018, pp. 144–155 (2018)
12. Sato, M., Hirano, M., Tanaka, Y., Sekiguchi, S.: Omnirpc: A grid RPC facility for cluster and global computing in openmp. In: OpenMP Shared Memory Parallel Programming, International Workshop on OpenMP Applications and Tools, WOMPAT 2001, West Lafayette, July 30–31, 2001 Proceedings, pp. 130–136 (2001)

www.dbooks.org

13. Tsuji, M., Sato, M., Hugues, M., Petiton, S.: Multiple-SPMD programming environment based on pgas and workflow toward post-petascale computing. In: Proceedings of 42nd International Conference on Parallel Processing, ICPP, pp. 480–485. IEEE, Piscataway (2013). https://doi.org/10.1109/ICPP.2013.58
14. Wu, X., Petiton, S.G.: A distributed and parallel asynchronous unite and conquer method to solve large scale non-hermitian linear systems. In: Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region, HPC Asia 2018, Chiyoda, January 28–31, 2018, pp. 36–46. ACM, New York (2018). https://doi.org/10.1145/3149457.3154481

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



TerraNeo—Mantle Convection Beyond a Trillion Degrees of Freedom



Simon Bauer, Hans-Peter Bunge, Daniel Drzisga, Siavash Ghelichkhan, Markus Huber, Nils Kohl, Marcus Mohr, Ulrich Rüde, Dominik Thönnes, and Barbara Wohlmuth

Abstract Simulation of mantle convection on planetary scales is considered a grand-challenge application even in the exascale era. The reason being the enormous spatial and temporal scales that must be resolved in the computation as well as the complexities of realistic models and the large parameter uncertainties that need to be handled by advanced numerical methods. This contribution reports on the TerraNeo project which delivered novel matrix-free geometric multigrid solvers for the Stokes system that forms the core of mantle convection models. In TerraNeo the hierarchical hybrid grids paradigm was employed to demonstrate that scalability can be achieved when solving the Stokes system with more than ten trillion ($1.1 \cdot 10^{13}$) degrees of freedom even on present-day peta-scale supercomputers. Novel concepts

S. Bauer · H.-P. Bunge
Ludwig-Maximilians-Universität München, Munich, Germany

D. Drzisga · S. Ghelichkhan · M. Huber
Technical University of Munich, München, Germany

N. Kohl
University of Erlangen–Nuremberg, Erlangen, Germany
e-mail: nils.kohl@fau.de

M. Mohr
Ludwig-Maximilians-Universität München, Munich, Germany
e-mail: marcus.mohr@lmu.de

U. Rüde (✉)
University of Erlangen–Nuremberg, Erlangen, Germany

Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique (CERFACS), Toulouse, France
e-mail: ulrich.ruede@fau.de

D. Thönnes
University of Erlangen–Nuremberg, Erlangen, Germany

B. Wohlmuth
Technical University of Munich, München, Germany
e-mail: barbara.wohlmuth@ma.tum.de

© The Author(s) 2020

H.-J. Bungartz et al. (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*, Lecture Notes in Computational Science and Engineering 136,
https://doi.org/10.1007/978-3-030-47956-5_19

www.dbooks.org

were developed to ensure resilience of algorithms even in case of hard faults and new scheduling algorithms proposed for ensemble runs arising in Multilevel Monte Carlo algorithms for uncertainty quantification. The prototype framework was used to investigate geodynamic questions such as high velocity asthenospheric channels and dynamic topography and to perform adjoint inversions. We also describe the redesign of our software to support more advanced discretizations, adaptivity, and highly asynchronous execution while ensuring sustainability and flexibility for future extensions.

1 Introduction and Motivation

Geodynamics

Mantle convection is a critical component of the Earth system. Although composed of rocks, the Earth's mantle behaves like a highly viscous fluid on geological time-scales. Its motion is driven by internal heating due to radioactive decay and by substantial heating from below. The latter stems from the release of primordial heat stored in the core from the time of the planet's accretion. The mantle convection currents are largely responsible for many of Earth's surface tectonic features, forming mountain chains and oceanic trenches through plate tectonics, and contributing significantly to the accumulation of stresses released in inter-plate earthquakes. Hence, a thorough quantitative understanding of mantle convection is indispensible to gain further insight into these processes. However, besides such sort of fundamental questions, mantle convection does also have direct influence on some societal and commercial issues. Viscous stresses caused by up- and downwellings in the mantle induce dynamic topography, i.e. they lead to elevation or depression of parts of Earth's surface. Reconstructing the latter and the associated sea-levels back in time is crucial for localisation of oil-reservoirs and the determination of future sea-level rises caused by climate change.

Although the basic equations for describing the mantle convection process are not in question, resulting from the force balance between viscous and buoyancy forces and conservation of mass and energy, key system parameters, such as the buoyancies and viscosities, remain poorly known. In particular the rheology of the mantle, which is a fundamental input parameter for geodynamic models, is not well known. Studies based on modeling the geoid e.g. [87], the convective planform e.g. [21], glacial isostatic adjustment e.g. [72, 80], true polar wander e.g. [82, 86, 89] and plate motion changes e.g. [57] consistently point to the need for a significant viscosity increase between the upper and the lower mantle. But the precise form of the viscosity profile remains uncertain. Commonly the viscosity profiles display a peak in the mid lower mantle [81, 85], or involve a rheology contrast located around 1000 km depth [93], or favor an asthenosphere with a strong viscosity reduction to achieve high flow velocities and stress amplification in the sublithospheric mantle [51, 102]. Geodynamic arguments on the uncertainties and trade-offs in the viscosity profile of the upper mantle have been summarized recently by Richards and Lenardic [88].

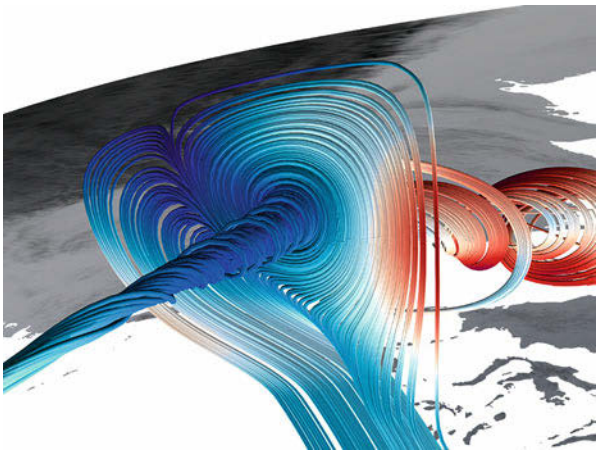


Fig. 1 Velocity flowlines under the Himalaya mountain range; snapshot from a global convection model simulated with hierarchical hybrid grids

Since the mantle is not directly accessible to observations and laboratory experiments can hardly reproduce the relevant temperatures, pressures, and time-scales, computer simulation is vital for studying mantle convection. An example is given in Fig. 1. One of the challenges is the enormous spatial scales that must be resolved. While the mantle has a thickness of roughly 3000 km, important features, such as subducting slabs might only have a width of several kilometers. Additionally, realistic simulations will require a very large number of time-steps. Furthermore, fully realistic models of the mantle must deal with many complexities such as non-linear viscosity models, phase transitions, or the treatment of the non-constant hydrostatic background density of the mantle, see e.g. [50].

TERRANEO

TERRANEO[1] aims to design and realize a new community software framework for extreme scale Earth Mantle simulations. We have a special constellation where groups from Geophysics, Numerical Mathematics, and High Performance Computing collaborate towards a unique co-design effort. The first 3-year funding period has already been summarized in [7]. Initially, the team had focussed on fundamental mathematical questions together with general scalability and performance issues, as documented in [38, 40–43]. In particular it could be shown that even with peta-scale class machines, computations are possible that resolve the global Earth Mantle with about 1 km resolution, requiring the solution of indefinite sparse linear systems with more than 10^{12} degrees of freedom. These very large computations and scaling experiments were performed with a prototype software that was implemented using the pre-existing hierarchical hybrid grids (HHG) multigrid library [12, 13]. This step was necessary to gain experience with parallel multigrid solvers for

[1] http://terraneo.fau.de.

www.dbooks.org

indefinite problems and the specific difficulties arising in Earth Mantle models. While this approach led to quick first results, the prototype character of HHG implied several fundamental limitations. Therefore, the second funding period is being used for a fundamental redesign of HHG under the new acronym HyTeG for Hybrid Tetrahedral Grids. The goal is to leverage the lessons learned with HHG for the design of a new, extremely fast, but also flexible, extensible, and sustainable Geophysics software framework.

Additionally, research on several exascale topics was conducted in both funding periods. These include resilience, inverse problems, and uncertainty quantification. Detailed results on these topics, as well as on the new HyTeG software architecture, will be reported in the following sections of this report.

Multigrid Methods

Multigrid methods belong to the set of fastest solvers for sparse linear systems that arise from the discretization of PDEs and are, thus, of central importance for exascale research. Their invention and their popularization e.g. in [17] constitute one of the major breakthroughs in numerical mathematics and computational science. To illustrate their relevance for exascale computing, we summarize here a thought experiment from [90].

Assuming that systems with $N = 10^{12}$, i.e. a trillion degrees of freedom (DoF) would be solved by classical Gaussian elimination, this would require the astronomical number of operations of $2/3N^3 = 2/3 \times 10^{36}$. Better elimination based algorithms can exploit the sparse matrix structure. A typical method of this class, such as *nested dissection* would still require around 10^{24} operations. For such a work load, a modern, fast PC with a performance of 100 GFlops would need more than 100,000 years of compute time. Note that solving one such system is still only worth one time step of many. This line of thought may be seen as just another argument why Earth Mantle Convection is a grand challenge problem and why e.g. the authors of [23] write:

> A uniform discretization of the mantle at for instance 1 km resolution would result in meshes with nearly a trillion elements, which is far beyond the capacity of the largest available supercomputers.

It is important to realize that parallel computing alone does not solve the problem. Even the fastest German supercomputer, currently SuperMUC-NG[2], would still need longer than a year of compute time (if it did not run out of memory before) to execute 10^{24} operations.

In the TerraNeo project we have demonstrated, and we will report below, that a well designed massively parallel multigrid method is indeed capable of solving such large systems with a trillion unknowns in compute times in the order of seconds. This is fundamentally based on their fast convergence, for many types of problems, that leads to asymptotically optimal complexity solvers, when it is combined with a nested iteration in the form of a so called full multigrid method [17, 41]. Based on this, it comes as little surprise that multigrid methods are employed in several of the

[2] https://doku.lrz.de/display/PUBLIC/SuperMUC-NG.

SPPEXA projects, such as e.g. [3, 4, 24, 69, 75] with excellent success for a wide variety of applications.

In the literature on multigrid methods of the past two decades, much attention has been given to so called algebraic multigrid methods (AMG) that do not operate on a mesh structure, but that attempt to construct the necessary hierarchy based on analyzing the sparse system matrix. These methods can exhibit excellent parallel scalability [3] and are often used as components within other parallel solvers, such as domain decomposition methods [62]. AMG methods have several advantages, most importantly that they can be interfaced to an application via classical sparse matrix technology. They can also save the application developer from worrying about the necessary solution process and many numerical analysts who design novel discretizations have taken this perspective. However, AMG methods work only well for certain types of systems, and consequently users of these new discretization techniques now find themselves being trapped with linear systems for which no efficient parallel algorithms are available, yet. Few methods devised in the applied mathematics community can demonstrate even solving systems with 10^9 degrees of freedom, falling three or four orders behind of what we can demonstrate here.

Additionally, the convenience of algebraic multigrid also comes at the price of a loss in performance. Algebraic multigrid methods have only a limited scope of applicability, and additionally they often lose an order of magnitude in performance in their expensive setup phase. Of course they are inherently also not *matrix-free*. As we will discuss below, matrix-free methods are essential to reach maximal performance. Geometric multigrid methods can be realized as matrix-free algorithms, potentially leading to another order of magnitude in performance improvement. For this reason we have invested heavily in new matrix-free methods [8, 9, 11], similar to other SPPEXA projects [5, 67].

The price of using geometric multigrid lies in the more complex algorithms which often have to be tailored carefully to each specific application, and the significantly more complex interfaces that are needed between the other software components and the solver. This in turn creates the need for new software technologies as are being developed in the HYTEG framework.

2 Basic Ideas and Concepts

As mentioned above the mathematical-physical model of mantle convection is based on the force balance between viscous and buoyancy forces and conservation of mass and energy. The resulting general system of equations is e.g. given in [7, 105]. Expressing these in terms of the central quantities of interest, i.e. velocity, pressure and temperature results in a coupled system composed of a stationary Stokes component and a time-dependent equation for the temperature. The former constitutes the most expensive part in these kinds of simulations and we will, thus, concentrate on this aspect mostly in the following. In the case of the Boussinesq

www.dbooks.org

Table 1 Physical quantities and their representing symbols

R_0	Earth's radius	ρ_0	Reference density	g	Gravitational acceleration
η	Dynamic viscosity	α	Thermal expansivity	κ	Heat conductivity
ΔT	Temperature difference between core-mantle-boundary and surface				

approximation the Stokes part is given in non-dimensional form by

$$-\mathrm{div}\big(\eta(\nabla \mathbf{u} + (\nabla \mathbf{u})^\top)\big) + \nabla p = -Ra\, T\, \mathbf{e}_r \tag{1}$$

$$\mathrm{div}(\mathbf{u}) = 0 \tag{2}$$

with dimensionless velocity $\mathbf{u}$, pressure p and the unit vector in radial direction $\mathbf{e}_r$. The Rayleigh number Ra describes the vigor of the convection and is given by

$$Ra = \frac{\alpha g \rho_0 \Delta T R_0^3}{\kappa \eta_r} \quad .$$

A description of the physical quantities represented by the symbols is given in Table 1. The finite element discretization that we consider for the stationary equations is backed by the concept of HHG. Initially, the domain is partitioned into an unstructured mesh of tetrahedra. In a second step, each tetrahedron is iteratively, uniformly refined in a way that a block-structured, tetrahedral mesh is created [12, 13, 40]. This refinement strategy results in a grid hierarchy $\mathcal{T} := \{\mathcal{T}_\ell, \ell = 0, 1, \ldots, L\}$ that allows for a canonical implementation of geometric multigrid methods.

The computational domain is distributed to the parallel processes by means of the unstructured coarse grid elements. To realize efficient communication procedures, the mesh is enhanced by so-called *interface primitives*. For each shared face, edge, and vertex an additional macro-primitive is allocated in the data structure. Together with the macro-tetrahedra, each primitive is assigned to one parallel process. Starting from a two times refined mesh, each macro-primitive has at least one inner vertex.

We employ conforming finite elements on the domain Ω and define the function spaces of globally continuous, piecewise polynomial functions with polynomial degree d on level ℓ as

$$S_\ell^d(\Omega) := \{v \in C(\overline{\Omega}) : v|_T \in P_d(T), \forall T \in \mathcal{T}_\ell\}.$$

For the majority of our experiments and simulations we implement a stabilized P1-P1 discretization for the Stokes equation, using the velocity and pressure finite element spaces

$$\mathbf{V}_\ell \times Q_\ell := \left[S_\ell^1(\Omega) \cap H_0^1(\Omega)\right]^3 \times S_\ell^1(\Omega) \cap L_0^2(\Omega).$$

Here, $L^2_0(\Omega) := \{q \in L^2(\Omega) : \langle q, 1\rangle_\Omega = 0\}$ with $\langle\cdot,\cdot\rangle_\Omega$ being the inner product in $L^2(\Omega)$. The P1-P1 element pairing does not fulfill the LBB-condition and we employ a PSPG stabilization [19, 33]. It follows the standard weak formulation for the discretized Stokes problem: find $(\mathbf{u}_\ell, p_\ell) \in \mathbf{V}_\ell \times Q_\ell$ such that

$$\begin{aligned} a(\mathbf{u}_\ell, \mathbf{v}_\ell) + b(\mathbf{v}_\ell, p_\ell) &= f(\mathbf{v}_\ell) \qquad \forall\, \mathbf{v}_\ell \in \mathbf{V}_\ell, \\ b(\mathbf{u}_\ell, q_\ell) - c_\ell(q_\ell, p_\ell) &= g_\ell(q_\ell) \qquad \forall\, q_\ell \in Q_\ell, \end{aligned} \tag{3}$$

where

$$\begin{aligned} a(\mathbf{u}, \mathbf{v}) &:= 2\langle \eta D(\mathbf{u}), D(\mathbf{v})\rangle_\Omega \\ b(\mathbf{u}, q) &:= -\langle \operatorname{div} \mathbf{u}, q\rangle_\Omega \\ f(\mathbf{v}) &:= \langle \mathbf{f}, \mathbf{v}\rangle_\Omega, \end{aligned}$$

for all $\mathbf{u}, \mathbf{v} \in H^1_0(\Omega)^3$ and $q \in L^2_0(\Omega)$. The stabilization terms are defined as

$$\begin{aligned} c_\ell(q_\ell, p_\ell) &:= \sum_{T\in\mathcal{T}_\ell} \frac{1}{12}\Big(\int_T dx\Big)^{1/3} \langle \nabla p_\ell, \nabla q_\ell\rangle_T \\ g_\ell(q_\ell) &:= -\sum_{T\in\mathcal{T}_\ell} \frac{1}{12}\Big(\int_T dx\Big)^{1/3} \langle \mathbf{f}, \nabla q_\ell\rangle_T . \end{aligned}$$

The discrete formulation of (3) can then be expressed as the system of linear equations

$$\mathcal{K}\begin{pmatrix}\mathbf{u}\\ \mathbf{p}\end{pmatrix} := \begin{pmatrix} A & B^\top \\ B & -C\end{pmatrix}\begin{pmatrix}\mathbf{u}\\ \mathbf{p}\end{pmatrix} = \begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix}. \tag{4}$$

A description of the general form of the temperature equation and our work on its discretization and time-stepping can be found in [7].

3 Summary of Project Results

3.1 Efficiency of Solvers and Software

A rigorous quantitative performance analysis lies at the heart of systematic research in large scale scientific computing. The systematic analysis of performance is central to the research agenda of computational science [91], since it defines how successful computational models are. In this sense, performance analysis here means more than measuring the speedup of a parallel solver or studying its weak and strong scaling

www.dbooks.org

properties. Beyond this, modern scientific computing must attempt to quantify numerical cost in metrics that are independent of the run-time of an arbitrary reference implementation. Most importantly, the numerical cost must be set in relation to what is numerically achieved, i.e. the accuracy that is delivered by a specific computation.

As one step in this direction, we extend Achi Brandt's notion of *textbook multigrid efficiency* (TME) to massively parallel algorithms in [41]. Using the finite element based prototype multigrid implementation of the HHG library, we have employed the TME paradigm for scalar linear equations with constant and varying coefficients as well as to linear systems with saddle-point structure. A core step is then to extend the idea of TME to the parallel setting in a way that is adapted to the parallel architecture under consideration. To this end, we develop a new characterization of a work unit (WU) in an architecture-aware fashion by taking into account modern performance modeling techniques, in particular the standard Roofline model and the more advanced ECM model. The newly introduced parallel TME measure is studied in [41] with large-scale computations and for solving problems with up to 200 billion unknowns.

3.2 Reducing Complexity in Models and Algorithms

When striving for optimal efficiency, it is essential to choose models and discretization schemes whose incurred computational cost is as low as possible. Within TERRANEO we have therefore analyzed and improved the discretization schemes under study. A restricting factor was initially that only simple conforming discretizations are feasible in the prototype HHG library. E.g. discontinuous Galerkin discretizations, as in [6, 69], were not yet feasible in HHG. In particular, buoyancy-driven flow models thus demand a careful treatment of the mass-balance equation to avoid spurious source and sink terms in the non-linear coupling between flow and transport. In the context of finite-elements, it is therefore commonly proposed to employ sufficiently rich pressure spaces, containing piecewise constant shape functions to obtain local or even strong mass-conservation. In three-dimensional computations, this usually requires nonconforming approaches, special meshes or higher order velocities, which make these schemes prohibitively expensive for some applications and complicate the implementation into legacy code. In [39], we propose and analyze a lean and conservatively coupled scheme based on standard stabilized linear equal-order finite elements for the Stokes part and vertex-centered finite volumes for the energy equation. We show that in a weak mass-balance it is possible to recover exact conservation properties by a local flux-correction which can be computed efficiently on the control volume boundaries of the transport mesh. Furthermore, we discuss implementation aspects and demonstrate the effectiveness of the flux-correction by different two- and three-dimensional examples which are motivated by geophysical applications in [101].

In the special case of constant viscosity the Stokes system can be cast into different formulations by exploiting the incompressibility constraint. For instance the strain in the weak formulation can be replaced by the gradient to decouple the velocity components in the different coordinate directions. Thus the discretization of the simplified problem leads to fewer nonzero entries in the stiffness matrix. This is of particular interest in large scale simulations where a reduced memory footprint and accordingly reduced bandwidth requirement can help to significantly accelerate the computations. In the case of a piecewise constant viscosity, as it typically arises in multi-phase flows, or when the boundary conditions involve traction, the situation is more complex, and the cross derivatives in the original Stokes system must be treated with care. A naive application of the standard vectorial Laplacian results in a physically incorrect solution, while formulations based on the strain increase the computational effort everywhere, even when the inconsistencies arise only from an incorrect treatment in a small fraction of the computational domain. In [55], we present a new approach that is consistent with the strain-based formulation and preserves the decoupling advantages of the gradient-based formulation in iso-viscous subdomains. The modification is equivalent to locally changing the discretization stencils, hence the more expensive discretization is restricted to a lower dimensional interface, making the additional computational cost asymptotically negligible. We demonstrate the consistency and convergence properties of the new method and show that in a massively parallel setup, the multigrid solution of the resulting discrete systems is faster than for the classical strain-based formulation.

3.3 Stokes Solvers and Performance

3.3.1 Multigrid Approaches for the Stokes System

In this subsection we briefly review different classes of solvers for Stokes type systems involving a multigrid component. One characteristic feature is the indefinite structure and thus special care in the design of the solver is required. Numerical experiments and performance studies were originally performed with the HHG software and are thus mostly limited to stabilized conforming P1-P1 discretizations. We point here also to the possibility to generate such parallel solvers automatically and achieve similarly good scalability and efficiency results, as demonstrated in the ExaStencils project [70, 75]. A comprehensive summary of the supercomputers used for the simulations conducted in this project is listed in Table 2.

In TERRANEO three classes of solvers we consider are a preconditioned Krylov method for the indefinite system, a preconditioned Krylov method for the positive definite pressure based Schur complement and a monolithic (sometimes also called an *all-at-once*) multigrid solver for the indefinite system. While in the first two approaches the multigrid solver is only applied to the velocity component, the monolithic multigrid scheme works simultaneously on the velocity and the pressure

www.dbooks.org

Table 2 Characteristics of the different supercomputers used for simulations presented in this publication

	SuperMUC Phase 1 (Thin nodes)	SuperMUC Phase 2	Juqueen	Hazel Hen
Operation	2012–2018	2015–present	2012–2018	2015–present
# nodes	9216	3072	28,672	7712
CPU	Intel Sandy Bridge E5-2580 8 Core	Intel Haswell E5-2697v3 14 Core	IBM PowerPC A2 16 Core	Intel Haswell E5-2680v3 12 Core
CPU frequency (GHz)	2.7	2.6	1.6	2.5
# total cores	147,456	80,016	458,752	185,088
Interconnect	Infiniband FDR10	Infiniband FDR14	5D Torus	Aries
Total memory (TByte)	288	194	448	987
Linpack (PFlop/s)	2.897	2.814	5.0	7.42

Table 3 Exploring the limits of the monolithic multigrid solver: weak scaling on JUQUEEN

Nodes	Threads	DoFs	Iter	Time [s]	Time [s] w/o coarse grid solver
5	80	2.7×10^9	10	685.88	678.77
40	640	2.1×10^{10}	10	703.69	686.24
320	5120	1.2×10^{11}	10	741.86	709.88
2560	40,960	1.7×10^{12}	9	720.24	671.63
20,480	327,680	1.1×10^{13}	9	776.09	681.91

component. A systematic quantitative performance study of all three approaches on modern peta-scale systems is given in [43]. More than 750,000 parallel threads are used in the largest simulation runs. Our main finding is that the all-at-once multigrid scheme outperforms the two alternatives. It does not only show the smallest memory footprint, but also results in the shortest compute time. More than 10 trillion[3] unknowns ($> 10^{13}$) have been solved for on a Blue Gene/Q system, see Table 3.

For such huge systems a sophisticated matrix-free code design with minimized memory traffic is a must. Although the use of GMRES type Krylov methods for the indefinite system is quite popular due to its robustness, it is not an option for us. The non-optimal memory requirement and compute time per iteration before the next restart slows down the overall performance drastically and restricts the feasible system size. To test the flexibility of the solver, we consider not only a box as geometry but also a thick spherical shell motivated by our application, a pipe filled with spherical obstacles of different diameters and a simplified artery, see Fig. 2. The physical system under consideration then ranges from a thermo-mechanically coupled system to a non-linear viscosity model of Carreau type.

[3] W.r.t. the short scale naming system.

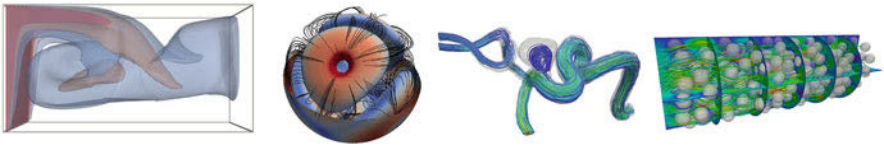


Fig. 2 Left to right: concentration in a lid driven cavity type setting; streamlines and speed in a benchmark problem; streamlines for a non-linear dynamic viscosity model and stationary flow in a pipe

3.3.2 Smoothers for Indefinite Systems

There are different strategies to construct smoothers for the coupled systems. Among the popular ones are Braess–Sarazin, Vanka, and Uzawa type approaches. While the first two types consider a saddle point problem either in a global or a local form as starting point, the Uzawa type scheme is based on a factorization of the saddle point system in lower and upper triangular matrices, see [16, 77, 98, 99, 104]. The advantage of the Uzawa version is clearly a smaller FLOP count and a reduced communication traffic compared to the two mentioned alternatives. Also in contrast to the Vanka smoother where local patches have to be considered it can be applied node-wise. Let us assume that A is symmetric and positive definite and $\hat{A}$ and $\hat{S}$ satisfy

$$\hat{A} \geq A, \quad \hat{S} \geq S := CBA^{-1}B^{\top}$$

then the following smoothing property holds

$$\|\mathcal{K}\mathcal{M}^{\nu}\| \leq \sqrt{2}\,\eta(\nu-1)\,\|\mathcal{D}\| \ ,$$

where ν is the number of smoothing steps, $\|\cdot\|$ a suitable operator norm and

$$\mathcal{M} := \mathrm{Id} - \mathcal{K}^{-1}\begin{pmatrix}\hat{A} & 0\\ B & \hat{S}\end{pmatrix}, \quad \mathcal{D} := \begin{pmatrix}\hat{A} & 0\\ 0 & \hat{S}\end{pmatrix}, \quad \eta(\nu) := \frac{1}{2^{\nu}}\begin{pmatrix}\nu\\ \lfloor(\nu-1)/2\rfloor\end{pmatrix}.$$

A mathematically rigorous proof is given in [29]. Following the abstract framework provided by Larin and Reusken [73], the following identity turns out to be of crucial importance

$$\mathcal{K}\mathcal{M}^{\nu} = M_1\mathcal{K}\mathcal{M}_s^{\nu-1}M_2$$

with suitable operators M_1, M_2 and $\mathcal{M}_s$ being the error propagation operator of the standard Uzawa. We note that in comparison to earlier theoretical results [95, 106, 107], we do not require symmetry of the Uzawa type smoother and thus

www.dbooks.org

Table 4 Number of iterations to reduce the residual by factor of 10^{-8} (Intel Xeon)

ν	4		6		8	
DoFs	Iter	Time [s]	Iter	Time [s]	Iter	Time [s]
1.4×10^3	13	0.10	9	0.07	7	0.06
1.4×10^4	12	0.21	9	0.18	7	0.15
1.2×10^5	12	0.61	8	0.51	6	0.44
1.0×10^6	11	2.44	8	2.33	6	2.16
8.2×10^6	11	14.54	8	14.58	6	14.03
6.6×10^7	11	102.66	7	92.09	6	101.90
5.3×10^8	10	700.38	7	693.75	6	769.34

the computational cost is reduced from two applications of the velocity smoother to one in each iteration. The achieved upper bound for the smoothing property is sufficient to guarantee level independent $\mathcal{W}$-cycle or variable $\mathcal{V}$-cycle results. However, in large scale applications $\mathcal{V}$-cycle schemes are much more attractive than $\mathcal{W}$-cycle based multigrid methods. The number of coarse solves per iteration is one in case of the $\mathcal{V}$-cycle but growths exponentially with the depth of the multigrid hierarchy in case of a $\mathcal{W}$-cycle. Unfortunately, there is no $\mathcal{V}$-cycle theory for this all-at-once multigrid method available, and moreover numerical experiments show that the convergence rate deteriorates for a fixed and level independent number of smoothing steps per level and an increased level count. As a compromise a mildly variable $\mathcal{V}$-cycle turns out to perform best. The HHG data structure and the matrix-free concept allows us, even on a standard workstation, to test the solver for systems with more than 100 millions of unknowns. The computations reported in Table 4 are performed on an Intel Xeon CPU E3-1226 v3, 3.30 GHz with 32 GB memory. For this first test only a single core is used and thus the timing does not reflect the parallel performance of the solver. For the pressure we use damped Gauss–Seidel applied to the operator C where the damping parameter is determined by a power iteration on a coarse level and not dependent on the mesh-size, but on the mesh-regularity.

While the all-at-once multigrid solver requires a special smoother, the Schur-complement formulation allows for a natural application of multigrid on the velocity component as part of an inner iteration in a preconditioned conjugate gradient method for the pressure. In case of an isoviscous flow the Schur complement is spectrally equivalent to the mass matrix and thus the condition number does not deteriorate with an increasing number of degrees of freedom. Due to its simple structure this iterative solver can be easily implemented by reusing standard software components and is thus suitable for a rigorous performance analysis. We do not report here in further detail, but an innovative performance analysis can be found in [40, 41]. Good scalability on more than half a million parallel threads is demonstrated in [42]. Together with the excellent node-level performance, this is essential for achieving high performance levels.

3.3.3 Multigrid Coarse Grid Solvers

Finally we report on results for agglomeration techniques. In the context of the matrix-free HHG implementation it is natural to use a preconditioned Krylov method as a coarse grid solver. However, in the context of very large systems this is a bottleneck since this coarse solver is non-optimal, and will thus ultimately become a limit to scalability. To improve the parallel efficiency for systems beyond a billion of unknowns, we proceed in two steps. In a first preprocessing step, the coarse grid problem is assembled in a classical fashion storing matrix entries in standard compressed row storage format. The system can then be solved by methods from the PETSc library, [79]. More specifically, a block preconditioned MINRES iteration is applied. For the velocity component we here use an AMG preconditioned conjugate gradient (CG) iteration and for the pressure a lumped mass-matrix approach. Here still the full parallel system is employed so that only small coarse grid subproblems are assigned to each parallel process. In the second step, we therefore propose to use agglomeration to reduce the number of parallel processes on the coarsest level and to reduce the communication overhead in each iteration. This becomes necessary when the ratio between communication time and compute time becomes unbalanced and when the overall coarse solve time is severely dominated by communication. In Table 5, we report on the parallel efficiency for a weak scaling study on JUQUEEN (IBM BlueGene/Q-System, 28 racks, 28,672 nodes, 458,752 cores). The number of unknowns ranges from approximately 80 million to over 100,000 million and the number of parallel threads is increased by a factor of more than 15,000. Over this large range, the parallel efficiency shows only a moderate deterioration and remains above 90% even in the largest run. This results mainly from the fact that for the largest run we reduce the number of processes on the coarsest level by a factor of eight. Using a master-slave agglomeration technique has the advantage of a short collecting and distribution phase.

In Fig. 3 we compare the naive non-optimal coarse solver with the PETSc based agglomeration strategy for the largest run, i.e. np $= 61{,}440$. We observe that with the naive approach roughly half of the compute time is spent in the coarse solve, although it has only a small fraction of the total number of unknowns. The situation is drastically different for the agglomeration technique in combination with the PETSc solver. Here only roughly 5% of the actual time to solve is spent in the coarse solver routines. A closer look at the ratio between MPI communication and operator computations exhibits that without agglomeration strategy a significant amount of

Table 5 Parallel efficiency for master-slave agglomeration technique and up to more than 50,000 parallel threads on JUQUEEN

np	DoF	Red.	T[s]	Coarse	Par. eff
30	8.3×10^7	1	16.284	0.043	1.00
120	3.3×10^8	1	16.426	0.050	0.99
960	2.6×10^9	1	17.084	0.171	0.95
7680	2.4×10^{10}	1	17.310	0.382	0.94
61,440	1.7×10^{11}	8	17.704	0.877	0.92

www.dbooks.org

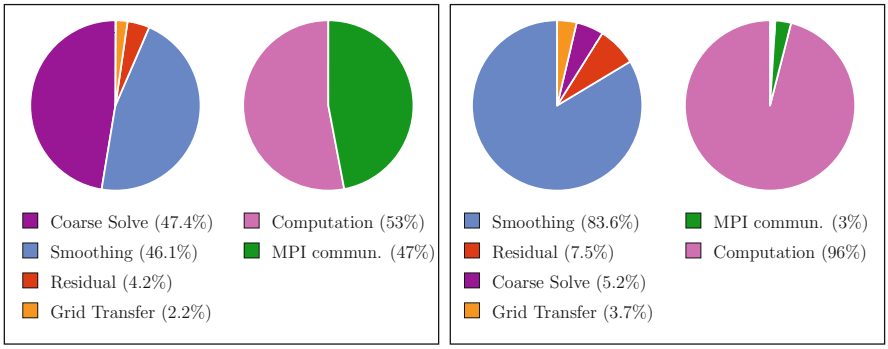


Fig. 3 Ratio of relative time to solve spent in the different multigrid components: a non-optimal Krylov based solver without agglomeration (left) and AMG preconditioned Krylov based solver with agglomeration

time is lost by simple communication. The situation is again drastically different for the agglomeration strategy. Here the communication overhead is reduced to less than 5%. We note that for a smaller system size and a moderate number of parallel threads, the difference in the time to solution for both approaches is negligible. For np $= 30$ the naive non-optimal coarse solver takes less than 10% of the total run-time and the communication load is less than 5% compared to the FLOP count. Moreover the savings in run-time in case of a PETSc coarse solver are limited due to the set-up phase. In conclusion, and somewhat contrary to conventional wisdom, we find that for up to ten million unknowns, a more sophisticated coarse solver does not pay off.

To show its generality, we also test the strategy on a different peta-scale system (Hazel Hen) and for a problem with a strongly varying discontinuous viscosity profile as it is relevant for geodynamical simulations, see Table 6. Here, we additionally exploit a block low-rank coarse grid solver (BLR) which uses low-rank compressions in an elimination method to devise an approximate solver [1]. The reduced parallel efficiency, compared to the setting before, does not result from the different architecture, nor the use of BLR, but mainly from the fact that one additional multigrid iteration is required.

Table 6 Parallel efficiency in a weak scaling experiment for a geodynamically relevant setting on Hazel Hen

	DOF	Iter	Time [s]			Time [s]		
Proc.	Fine		Total	Fine	BLR ϵ	Coarse	Ana. & fac.	Par. eff.
1920	2.10×10^{10}	15	78.1	77.9	10^{-3}	0.03	2.7	1.00
15,360	4.30×10^{10}	13	88.9	86.8	10^{-3}	0.22	25.0	0.93
43,200	1.70×10^{11}	14	95.5	87.0	10^{-8}	0.59	111.6	0.82

3.4 Multi-Level Monte Carlo

Due to errors in measurements, geophysical data are inevitably stochastic. Therefore, it is desirable to quantify these uncertainties when computing geophysical applications. A typical approach is to use ensemble simulations, i.e. running multiple computations with slight variations of perturbed data, and evaluating computational quantities of interest via Monte Carlo sampling. A naive sampling-based uncertainty quantification for 3D partial differential equations results in an extremely large computational complexity. More sophisticated approaches, such as multilevel Monte Carlo (MLMC), can reduce this complexity significantly. The performance can be further enhanced when the Monte Carlo sampling over several levels of mesh refinement is combined with a fast multigrid solver. In a parallel environment, however, sophisticated scheduling strategies are needed to exploit MLMC based on multigrid solvers. In [28], we explored the concurrent execution across the three layers of the MLMC method. Namely, parallelization across levels, across samples, and across the spatial grid.

An alternative way to classify these different parallel execution models is to consider the time-processor diagram, as illustrated in Fig. 4, where the scheduling of each sample Y_ℓ^i, with index $1 \leq i \leq N_\ell$ on level $0 \leq \ell \leq L$, is represented by a rectangular box with the height expressing the number of processors used. We call a parallel execution model *homogeneous bulk synchronous* if at any time in the processor diagram all tasks execute on the same level ℓ with the same number of processors. Otherwise we call it *heterogeneous bulk synchronous*.

The one-layer homogeneous strategy, as shown in Fig. 4 (left), offers no flexibility. The theoretical run-time is simply given by the sum of the time of all samples. Even if this method guarantees perfect load balancing, it will not lead to an optimal efficiency since on the coarser levels the scalability of the solver is typically significantly worse than on the finer levels. On the coarsest level we may even have less grid points than processors. Thus, we discard this approach from our list of possible options.

We, instead, consider the following two variants: Firstly, *sample synchronous homogeneous (SaSyHom)* where a synchronization step is imposed after each sequential step (see Fig. 5, left). Here statistical quantities can be updated after

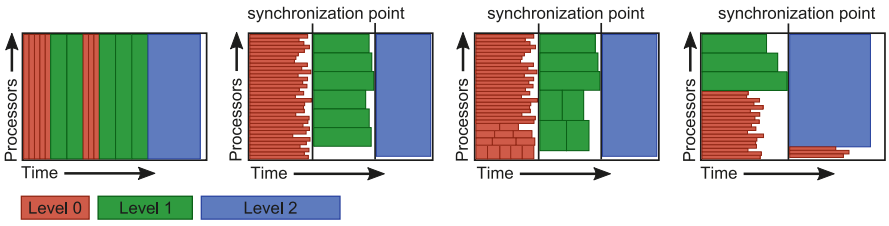


Fig. 4 From left to right: illustration of homogeneous (one-layer and two-layer) and of heterogeneous bulk synchronous strategies (two-layer and three-layer)

www.dbooks.org

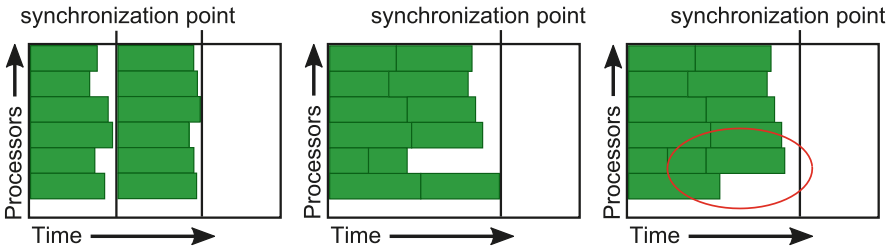


Fig. 5 Illustration of different homogeneous scheduling strategies. Left: sample synchronous homogeneous (SaSyHom); centre: level synchronous homogeneous (LeSyHom); right: dynamic level synchronous homogeneous (DyLeSyHom)

each synchronization point. Secondly, *level synchronous homogeneous (LeSyHom)*, where each block of processors executes all samples without any synchronization (see Fig. 5, center).

In case of large run-time variations of the samples, we additionally consider dynamic variants where samples can be assigned to processors dynamically. Figure 5 (right) illustrates the DyLeSyHom strategy. Here it is essential that not all processor blocks execute the same number of sequential steps.

We compare static strategies with their dynamic variants in the case of large run-time variations on three MLMC levels, i.e. $L = 2$, with a fine grid resolution of about $1.6 \cdot 10^7$ mesh nodes. We assume a lognormal random coefficient, but in order to increase the run-time variation, we choose a greater variance $\sigma^2 = 4.0$ and $\lambda = 0.5$. As quantity of interest we employ the PDE solution u evaluated at the point $x = (0.5, 0.5, 0.5)$. All samples are computed with a FMG-2V(4,4) cycle with up to a hundred additional V(4,4) cycles until a relative residuum of 10^{-10}. Pre-computed variance estimates lead to an a priori strategy with $(N_\ell)_{\ell=0,1,2} = (27{,}151, 10{,}765, 3792)$ samples per level. In this scenario, 8192 processors were available for the scheduler. In Table 7 we compare the required time of the LeSyHom strategy in the static and dynamic variant and see that the dynamic strategy is 6% faster than its static counterpart.

The largest uncertainty quantification scenario we considered, involved finest grids with almost 70 billion degrees of freedom, and a total number of samples beyond 10,000, most of which are computed on coarser levels. This computation was executed on the JUQUEEN supercomputer using more than 132,000 processor cores in an excellent overall compute time of about 1.5 h.

Table 7 Comparison of static and dynamic scheduling strategies

Level	LeSyHom time [s]	DyLeSyHom time [s]	Ratio
0	500	460	0.92
1	1512	1347	0.89
2	5885	5596	0.95
Total	**7897**	**7403**	**0.94**

3.5 *Inverse Problem and Adjoint Computations*

The adjoint method is a powerful technique that allows the computation of sensitivities (Fréchet derivatives) with respect to model parameters. It solves inverse problems where analytical solutions are not available or the cost to solve the associated forward problem many times is prohibitively high. In geodynamics it has been applied to the inverse problem of mantle convection—i.e., to restore past mantle flow e.g. [22], where it finds an optimal initial flow state that evolves into the present-day state of Earth's mantle. In doing so, the adjoint method has the potential to link together diverse observations and theoretical expectations from seismology, geology, mineral physics, paleomagnetism and fluid dynamics, greatly enhancing our understanding of the solid Earth system.

Adjoint equations for mantle flow restoration have been derived for incompressible [22, 52, 59], compressible [35] and thermo-chemical mantle flow [36], and the uniqueness properties of the inverse problem have been related explicitly to the tangential component of the surface velocity field of a mantle convection model [25]. So knowledge of the latter is essential to assure convergence [100] and to obtain a small null space for the restored flow history [52]. To reduce the computational cost, we use a two-scale time step size predictor-corrector strategy to couple between Stokes and temperature similar to the one discussed in [46].

The framework was used to implement the adjoint technique. Specifically, we minimized the misfit functional

$$\chi(T) := \frac{1}{2} \int_{\Omega} \int_{[t_0,t_1]} \Big(T(T_0, \mathbf{x}, t) - T_E(\mathbf{x}) \Big)^2 \delta(t - t_1) \, \mathrm{d}t \, \mathrm{d}x \tag{5}$$

that relates to the squared difference of the geodynamic model temperature and the thermal heterogeneity distribution T_E for the final state at $t = t_1$. The latter can be inferred e.g., from seismology and considerations of mantle mineralogy. We used a first order optimization approach to update the initial temperature T_0 in the descent direction ϕ in a process described by

$$T_0^{i+1}(\mathbf{x}) = T_0^i(\mathbf{x}) - \beta \phi(\mathbf{x}, t_0) \quad \text{for } i = 0, 1, \ldots ,$$

where β controls the correction in the descent direction and has to be sufficiently small [84]. In order to reduce the computational cost of deriving the step length, we found that fixing $\beta = 0.4$ yields sufficient results in our experiments, in agreement with [22]. The descent direction $\phi(\mathbf{x}, t_0)$ is obtained by solving the adjoint equations, which are given by Bunge et al. [22]. Solving the adjoint equations requires velocity and temperature states from the forward problem. We checkpointed these states in the forward problem such that they could be read for the adjoint iteration. In our exploration of the inverse problem, we used a similar setup as presented in [23]. We considered a $45° \times 45°$ part of a spherical shell domain and discretized it by a tetrahedral mesh. Applying four uniform refinement

www.dbooks.org

steps, the mesh has a resolution of $2.7 \cdot 10^5$ grid points. The initial temperature field given by

$$T(\mathbf{x}) = T_0 + \exp\left(-\frac{1}{\sigma^2}\|\mathbf{x} - \mathbf{x}_0\|^2\right),$$

where $\sigma = 1/20$ determines the extent of the anomaly and $\mathbf{x}_0$ denotes the center. Note that in our setup the latter is situated $2/9\,D$ below the core-mantle boundary, where D denotes relative mantle thickness, and thus lower than in [23].

3.5.1 Twin Experiment

To verify our implementation, we considered a so called *twin experiment*. In this setup a reference flow history is generated from the forward model. The final state of this reference flow is used as the terminal state that drives the geodynamic inverse problem. The reconstructed flow history is then compared against the known reference history to assess the quality of the inversion. Thus, while in a realistic geophysical setting only the final state T_E is known, in the twin experiments, the true (unreconstructed) initial T_I and final state T_E are given from the reference twin. This allows us to study the convergence of initial and final states. In addition to the misfit error (5) at the final time, we define the misfit error at the initial time by

$$m_I(T) \coloneqq \frac{1}{2}\int_\Omega \int_{[t_0,t_1]} \Big(T(\mathbf{x},t) - T_I(\mathbf{x})\Big)^2 \delta(t - t_0)\,\mathrm{d}t\,\mathrm{d}x$$

In Fig. 6, we present the initial/final temperature (left/right) from the reference twin. Figure 7 shows isosurfaces of the reconstructed initial (top row) and final (bottom row) temperature after adjoint iterations $i = 0, 5, 10$. For the sake of simplicity, the first guess for the unknown initial temperature is taken from the final state of the reference twin. Therefore, the reconstructed final temperature for $i = 0$ is a plume that has evolved much farther than the final reference state. After solving the inverse problem with 10 iterations, the error at the final state is reduced by more than one order of magnitude, while the error at the initial state is reduced by a



Fig. 6 Left: initial temperature state, right: final temperature state of the reference twin

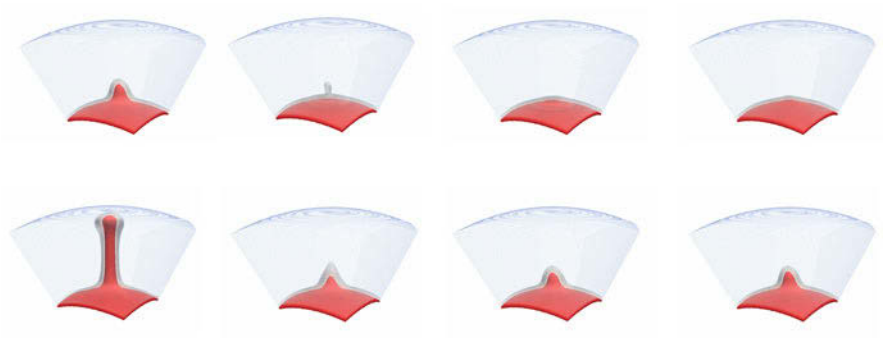


Fig. 7 From left to right: reconstruction of initial (top row) and final (bottom row) state: $i = 0, 5, 10$ iterations, very right: reference states

factor of 3, and we observe a good agreement between reconstructed and reference temperatures.

3.6 Matrix-Free Algorithms

Matrix-free approaches are an essential component for ultra-high resolution finite-element simulations. The reason for this is twofold. The classical finite-element workflow of assembling the global system matrix from the local element contributions and then solving the resulting linear system of equations with algorithms implemented in matrix-vector fashion leads to enormous data traffic between main memory and computational units (CPU cores). This constitutes a severe performance bottleneck. Another aspect is that the cost for holding the matrix in main memory becomes prohibitive, too. Note that for our simulations with 10^{13} DoFs, see [43] and Sect. 3.3 the solution vector alone requires 80 TByte of memory. Thus, matrix-free methods, which do not assemble the global matrix, but only provide the means to evaluate an associated matrix–vector product receive increasing attention, see e.g. [5, 20, 66–68, 78, 92]. An overview on the history and different approaches can e.g. be found in [9].

We note that the concept of hierarchical hybrid grids, from its inception on, was designed to be matrix-free. Based on its original premise that the macro mesh resolves both, geometry and material parameters, only a single discretization stencil had to be computed and stored for each macro primitive, as opposed to one for every fine mesh node, which corresponds to one row of the global matrix, [13]. For the case of locally varying material parameters, such as the viscosity in mantle convection, this could be extended, using classical local element matrix based approaches, see [9, 14, 37]. However, this approach does not carry over to the case of non-polyhedral domains and/or higher order elements. Thus, we developed alternative and also more efficient approaches.

www.dbooks.org

3.6.1 Matrix-Free Approaches Based on Surrogate Polynomials

In our HHG and HyTeG frameworks the coupling between DoFs is considered in the form of a stencil like in classical finite differences. Due to the local regularity of the refined block-structured mesh the stencil pattern is invariant across an individual macro primitive. For example in the case of P_1 elements and a scalar equation in 3D we obtain a 15-point stencil for all nodes within a volume primitive or on a face primitive. However, in the case of a curved domain and/or locally variable material parameters the stencil weights are non-constant over a macro primitive. These weights can be computed on-the-fly when they are needed to apply the stencil locally, e.g. during a smoothing step, by standard quadrature. Note that at least for P_1 elements this is theoretically still faster than the more sophisticated approach of fusing quadrature and application of the local element-matrix, see [66]. However, it is significantly slower than the original one-stencil-per-primitive scenario. Thus, in [8] we introduce a new two-scale approach that employs surrogate polynomials to approximate the stencil weights. The idea, in a nutshell, is to consider the individual stencil weights as functions on each primitive. These functions can be sampled, in a setup phase, on a certain number of sampling points, typically for all nodes of the primitive on a coarser level of the mesh hierarchy. Then an approximating polynomial is constructed for the weight by determining the polynomial coefficients through a standard least-squares fit. Whenever the stencil weight is needed the associated polynomial is then evaluated. Performance-wise this raises two questions. How much memory is required for storing the polynomial coefficients and what is the cost for evaluating the polynomial, as opposed to that of on-the-fly quadrature. The answer to the first question is that the memory footprint even in 3D is not too large for polynomials of moderate degree. Consider as an example a trivariate polynomial of degree five. It has 56 coefficients. Representing a 15-point stencil by such polynomials for all nodes of a volume primitive, thus, requires only 6720 bytes. Note also that the number of polynomials needed can be reduced by symmetry arguments and the zero row-sum property of consistent weak differential operators, see [8, 30]. Thanks to the logical structuredness of the mesh inside our volume and face primitives evaluation of a polynomial can be performed lexicographically along lines. It, thus, reduces to a 1D problem which can efficiently be executed using incremental updates based on the concept of divided differences, see [8, 30].

In [8] we conducted an extensive performance study of the new approach for the 3D Poisson problem on different curved domains which we connected to a polygonal domain by means of a mapping function. Thus, we are in fact solving a diffusion equation with a non-constant symmetric definite material tensor. As an example Fig. 8 shows results for a weak scaling experiment on SuperMUC. Here we consider a V(3,3) cycle and assign 16 macro elements to one core. On the finest grid there are 3.3×10^5 DoFs per volume primitive. We see that the surrogate operator approach (using quadratic polynomials) clearly outperforms the on-the-fly quadrature and is only about a factor two more expensive than the original one-stencil-per-primitive approach in HHG which cannot capture the domain curvature.

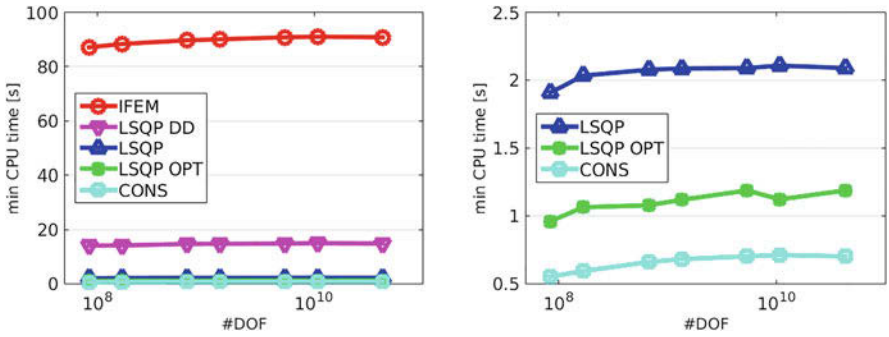


Fig. 8 Weak scaling behavior of a V(3, 3) cycle for different approaches: on-the-fly quadrature (IFEM), surrogate polynomials (LSQP), surrogate polynomials combined with double discretization (LSQP DD) and surrogate polynomials after node-level performance optimization (LSQP OPT); for comparison (CONS) gives times for the one-stencil-per-primitive approach that neglects curvature

Additionally, we study the influence of the approximation on the discretization error and the multigrid convergence rate. The former is influenced by the ratio between the mesh size of the macro mesh and the solution level, the sampling level and also the polynomial degree, while the latter is basically non-existent, see Table 8. In [30] we present a theoretical analysis of the consistency error of the surrogate operator approach and extend our numerical experiments to more challenging differential operators using problems from linear elasticity and nonlinear p-Laplacian diffusion.

In mantle convection models viscosity may vary significantly on a local scale, i.e. within a volume primitive. In this case, an approximation of stencil weights by low-order polynomials as in [8] will no longer be sufficient. One might try higher order polynomials as was implemented in HYTEG for the tests presented in [30]. Albeit this is currently only available in 2D. Alternatively, we devised a variant of our approach, see [10, 11], denoted in the following as LSQP$_e$. Here, instead of directly approximating the stencil weights, we replace the entries of the local element matrices by surrogate polynomials. The stencil weights are then computed

Table 8 Discretization error in the L^2-norm for IFEM and LSQP (with different polynomial degree q) and ratio of the asymptotic multigrid convergence rates for curved pipe domain (level $\ell = 0$ refers to an already twice refined mesh)

Level	e_{IFEM}	e_{LSQP}	$\rho_{\text{IFEM}}/\rho_{\text{LSQP}}$	e_{LSQP}	$\rho_{\text{IFEM}}/\rho_{\text{LSQP}}$
		$q = 2$		$q = 3$	
$\ell = 1$	7.50×10^{-5}	7.50×10^{-5}	1.00	7.50×10^{-5}	1.00
$\ell = 2$	1.86×10^{-5}	1.86×10^{-5}	1.00	1.86×10^{-5}	1.00
$\ell = 3$	4.64×10^{-6}	4.67×10^{-6}	1.00	4.64×10^{-6}	1.00
$\ell = 4$	1.16×10^{-6}	1.41×10^{-6}	1.00	1.16×10^{-6}	1.00
$\ell = 5$	2.89×10^{-7}	9.87×10^{-7}	1.00	3.10×10^{-7}	1.00

www.dbooks.org

Table 9 Weak scaling of the LSQP_e approach using ca. 2.3×10^7 DoFs per core; shown are: average run-time w/ and w/o coarse grid solver (c.g.) for one UMG cycle and no. of UMG iterations; values in brackets give no. of c.g. iterations (preconditioner/MINRES); parallel efficiency w.r.t. one UMG cycle is given for timings w/ and w/o c.g.; additionally the average time for a single residual application on the finest level is given

Islands	Cores	DoFs	Global resolution	# UMG V-cycles	Time UMG cycle		Parallel efficiency	Time residual
					w/ c.g	w/o c.g.		
1	5580	1.3×10^{11}	3.4 km	7 (50/150)	192 s	164 s	1.00 / 1.00	11.9 s
2	12,000	2.7×10^{11}	2.8 km	10 (100/150)	213 s	169 s	0.90 / 0.97	12.1 s
4	21,600	4.8×10^{11}	2.3 km	7 (50/250)	210 s	172 s	0.92 / 0.96	12.7 s
8	47,250	1.1×10^{12}	1.7 km	8 (50/350)	230 s	173 s	0.83 / 0.95	12.8 s

from these surrogate element matrices in the standard fashion for the P_1 case. This variant was used to perform the dynamic topography simulations of Sect. 3.10. As an example Table 9 shows weak scaling results on SuperMUC Phase 1 for a mantle convection model with a temperature-dependent viscosity and a global resolution of 1.7 km for the largest scenario. Run-times are for a V-cycle with an Uzawa smoother, stopping criterion is a residual reduction of five orders of magnitude. For complete details we refer to [11]. Note the parallel efficiency of 83% for the largest run, which could be further improved by using a more sophisticated coarse grid solver, see Sect. 3.3, than the one available at the time of the experiment.

3.6.2 A Stencil Scaling Approach for Accelerating Matrix-Free Finite Element Implementations

In [9] we present a novel approach to fast on-the-fly low order finite element assembly for scalar elliptic partial differential equations of Darcy type with variable coefficients optimized for matrix-free implementations. In this approach, we introduce a new operator that is obtained by scaling the reference operator, i.e. the stencil obtained from the constant coefficient case. Assuming sufficient regularity, an a priori analysis showed that solutions obtained by this approach are unique and have asymptotically optimal order convergence in the H^1-norm and the L^2-norm on hierarchical hybrid grids. These preliminary considerations motivate our novel approach to reduce the cost by recomputing the surrogate stencil entries for a matrix-free solver in a more efficient way.

To demonstrate the advantages of our novel scaling approach we consider, among other examples, a Poisson problem on a domain defined through a blending function. Particularly, we consider a half cylinder mantle with inner radius $r_1 = 0.8$ and outer radius $r_2 = 1.0$, height $z_1 = 4.0$ and with an angular coordinate between 0 and π as our physical domain Ω_{phy}. The cylinder mantle is additionally warped inwards by

$w(z) = 0.2 \sin(z\pi/z_1)$ in axial direction. The mapping $\Phi : \Omega_{\text{phy}} \to \Omega$ is given by

$$\Phi(x, y, z) = \begin{pmatrix} \sqrt{x^2 + y^2} + w(z) \\ \arccos\left(x/\sqrt{x^2+y^2}\right) \\ z \end{pmatrix}$$

with the reference domain $\Omega = (r_1, r_2) \times (0, \pi) \times (0, z_1)$. It follows for the mapping tensor K of the Poisson problem that

$$K = \frac{(D\Phi)(D\Phi)^\top}{|\det D\Phi|} = \sqrt{x^2 + y^2} \begin{pmatrix} w'(z)^2 + 1 & 0 & w'(z) \\ 0 & 1/(x^2+y^2) & 0 \\ w'(z) & 0 & 1 \end{pmatrix} .$$

Obviously, this tensor is symmetric and positive definite. In addition to the geometry blending, we use a variable material parameter $a(x, y, z) = 1 + z$. This yields the following PDE on the reference domain Ω: $-\text{div}(aK\nabla u) = f$. Additionally, the manufactured solution is chosen as

$$u(x, y, z) = \sin\left(\frac{x - r_1}{r_2 - r_1}\pi\right) \cos(4y) \exp(z/2) \quad .$$

For our numerical experiments, we employ a macro mesh composed of 9540 hexahedral blocks, where each block is further split into six tetrahedral elements. The resulting system is solved on SuperMUC Phase 2 using 14,310 compute cores, i.e., four macro elements are assigned per core. The largest system involves solving a linear system with $\mathcal{O}(10^{11})$ DoFs. We employ a multigrid solver with a V(3,3) cycle and the iterations are stopped when the residual has been reduced by a factor of 10^{-8}. In Table 10, we report on the resulting discretization error and its estimated order of convergence (eoc), the asymptotic multigrid convergence rate ρ, and the time-to-solution for different refinement levels ℓ. Note that we restricted the scaling approach in this test to volume and face primitives, which contain the vast majority of DoFs.

Table 10 Results for large scale 3D application with errors measured in the discrete L^2-norm

	Nodal integration				Scale Vol+Face				Rel.
DoFs	Error	eoc	ρ	tts [s]	Error	eoc	ρ	tts [s]	tts
4.7×10^6	2.43×10^{-4}	–	0.522	2.5	2.38×10^{-4}	–	0.522	2.0	0.80
3.8×10^7	6.00×10^{-5}	2.02	0.536	4.2	5.86×10^{-5}	2.02	0.536	2.6	0.61
3.1×10^8	1.49×10^{-5}	2.01	0.539	12.0	1.46×10^{-5}	2.01	0.539	4.5	0.37
2.5×10^9	3.72×10^{-6}	2.00	0.538	53.9	3.63×10^{-6}	2.00	0.538	15.3	0.28
2.0×10^{10}	9.28×10^{-7}	2.00	0.536	307.2	9.06×10^{-7}	2.00	0.536	88.9	0.29
1.6×10^{11}	2.32×10^{-7}	2.00	0.534	1822.2	2.26×10^{-7}	2.00	0.534	589.6	0.32

www.dbooks.org

These results demonstrate that the new scaling approach reproduces the discretization error, as is expected from our variational crime analysis [8, 9, 30]. Additionally, the multigrid convergence rate is not affected. For larger ℓ the runtime as compared to the nodal integration approach requires only about 30% of the time.

3.6.3 Stencil Scaling for Vector-Valued PDEs with Applications to Generalized Newtonian Fluids

Our target problem is vector-valued, thus, we expanded the scalar stencil scaling idea from Sect. 3.6.2 and developed a similar matrix-free approach for vector-valued PDEs [31]. The construction is again based on the use of hierarchical hybrid grids, the conceptual basis in the HHG and HyTeG [63] frameworks. Vector-valued second-order elliptic PDEs play an important role in mathematical modeling and arise e.g. in problems from elastostatics and fluid dynamics. Numerical experiments indicated that the idea of the scalar stencil scaling (denoted below as unphysical scaling) cannot directly be applied to these equations, because the standard finite-element solution cannot be reproduced, even in the case of linear coefficients. Thus, there is need of a modified stencil scaling method (denoted below as physical scaling) that is also suited for matrix-free finite element implementations on hierarchical hybrid grids. It turns out this vector-valued scaling requires computation of an additional correction term. While this makes it more complicated and expensive, compared to the scalar stencil scaling, it is able to reproduce the standard finite-element solutions, while requiring only a fraction of the time to obtain them. In the best scenario, we could observe a speedup of about 122% compared to standard on-the-fly integration. Our largest example involved solving a Stokes problem with 12,288 compute cores.

One of the examples studied is the scenario of a non-linear incompressible Stokes problem where the fluid is assumed to be of generalized Newtonian type, modeled by a shear-thinning Carreau model

$$\mu(\boldsymbol{u}) = \eta_\infty + (\eta_0 - \eta_\infty)\left(1 + \kappa|\varepsilon(u)|^2\right)^r .$$

The parameters employed are given in Table 11 in dimensionless form. The values stem from experimental results, cf. [34, Chapter II].

The computational domain Ω is depicted in Fig. 9. The channel has a length of 5 and a depth and height of 1. The kink starts at position $x = 1.5$. The domain is discretized by 14,208 tetrahedra on the coarsest level. The boundary

Table 11 Dimensionless parameters for the Carreau viscosity model

Parameter	η_0	η_∞	κ	r
Value	140.764	1.0	212.2	−0.325



Fig. 9 Velocity streamlines for the non-linear generalized Newtonian Stokes problem

$\partial\Omega$ is composed into Dirichlet and Neumann parts, i.e., $\partial\Omega = \Gamma_D \cup \Gamma_N$ with $\Gamma_D = \{(x, y, z) \in \partial\Omega \mid x < 5\}$ and $\Gamma_N = \partial\Omega \backslash \Gamma_D$. The volume force term $\boldsymbol{f}$, the external forces $\hat{\boldsymbol{t}}$, and the Dirichlet boundary term $\boldsymbol{g}$ are set to

$$\boldsymbol{f} = (0, 0, 100)^\top, \ \hat{\boldsymbol{t}} = (0, 0, 0)^\top, \text{ and } \boldsymbol{g} = 16\, y\,(1 - y)\, z\,(1 - z) \cdot (1, 0, 1)^\top.$$

We solve this non-linear system by applying an inexact fixed-point iteration where the underlying linear systems are only solved approximately to prevent over-solving.

In order to solve the systems, we employ the inexact Uzawa solver presented from [29] with variable $V(3, 3)$ cycles where 2 smoothing steps are added to each coarser refinement level which enforces convergence of the method.

In Table 12, we present the relative errors of the physical and unphysical scaling approaches along the line $\theta = [0, 5] \times \{0.5\} \times \{0.42\}$ in the supremum norm, computed on the 4 times refined grid. It can clearly be seen that the physical scaling yields significantly better results with errors smaller by three orders of magnitude. Further experiments indicate that the unphysical scaling does not converge to the standard finite element solution even after additional mesh refinements. Therefore, we conclude that the unphysical scaling is an inconsistent discretization for vector-valued problems.

Table 13 presents the relative time-to-solutions for the nodal integration and physical scaling measured on SuperMUC Phase 2. On the finest level, after 6 refinements, the relative time-to-solution of the physical scaling is at about 81%. Figure 9 shows the streamlines of the velocity within the computational domain computed with the physical scaling.

Table 12 Relative errors along the line θ in the supremum norm for different scaling approaches

Viscosity		Velocity	
Physical	Unphysical	Physical	Unphysical
8.88×10^{-4}	1.07×10^{-1}	5.65×10^{-5}	3.19×10^{-2}

www.dbooks.org

Table 13 Relative time-to-solution comparison of the nodal integration and physical scaling approach for the non-linear generalized Stokes problem

DoFs	Nodal integration tts [s]	Physical scaling tts [s]	Relative tts
4.69×10^6	309.10	364.97	1.18
3.82×10^7	361.90	412.10	1.14
3.08×10^8	895.18	719.55	0.80
2.47×10^9	3227.45	2626.13	0.81

3.7 Resilience

Extremely concurrent simulations may in the future require a software design that is resilient to faults of individual software and hardware components. The probability of faults in the underlying systems increases with the number of parallel processes. Especially long-running simulations may suffer from lower mean time between failures and restarting applications with run-times of several hours or days consumes vast amounts of resources.

Therefore, fault tolerance techniques have become a research topic as preparation for possibly unreliable future exascale systems. In the TERRANEO project we have so far mainly focused on hard faults. To cope with the failure of a core or node, we mainly distinguish between two categories of methods: one class relies on checkpointing techniques, where snapshots of the simulation are stored in regular intervals so that the state can be loaded upon failure. A second class are algorithm-based fault tolerance (ABFT) techniques, where (partially) lost data can be re-computed on-the-fly. Regarding large-scale simulations, checkpointing techniques often suffer from bad serial and parallel performance if data is written to disk. However, there are approaches that solely rely on distributed, in-memory checkpointing, which could be combined with compression techniques [71] that were also explored in a student project [74]. This can lead to a flexible, fast, and scalable resilience method [64].

In [53, 54] an alternative ABFT technique to provide resilience specifically for multigrid methods is developed. Given a faulty subdomain Ω_F, created when a processor (core or node) crashes, the global solution can be recovered by solving a recovery problem on Ω_F via a local multigrid iteration. After the recovery subproblem is solved, the global iteration continues.

A priori, it is not clear, how many iterations will be required in the faulty subdomain. In [54] a fixed number of V-cycle iterations is employed. This can lead to under– or over-solving in Ω_F. While under-solving leads to a poor recovery, over-solving inhibits the global convergence due to wrong values at the interface. The convergence behavior of both scenarios is illustrated in Fig. 10.

To address this issue, in [56] the recovery algorithm is enhanced by an adaptive control mechanism. Instead of solving the subproblem using a fixed number of iterations, a hierarchically weighted error estimator is introduced to define a stopping criterion for the faulty subdomain. The estimator is designed to be well-suited to extreme-scale parallel multigrid settings as they are employed in

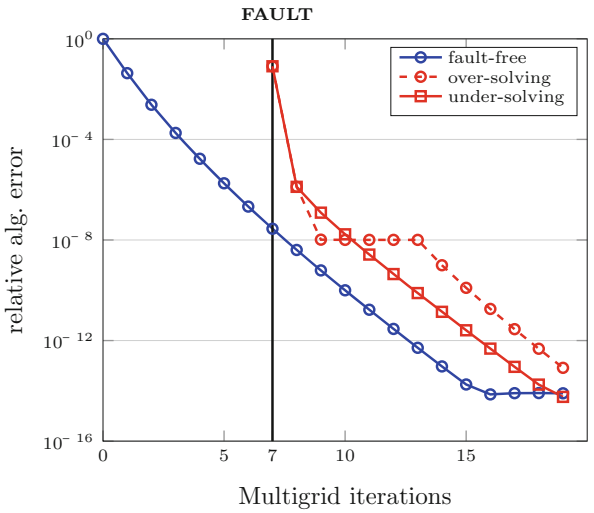


Fig. 10 Convergence behavior for over- and under-solving in the faulty subdomain

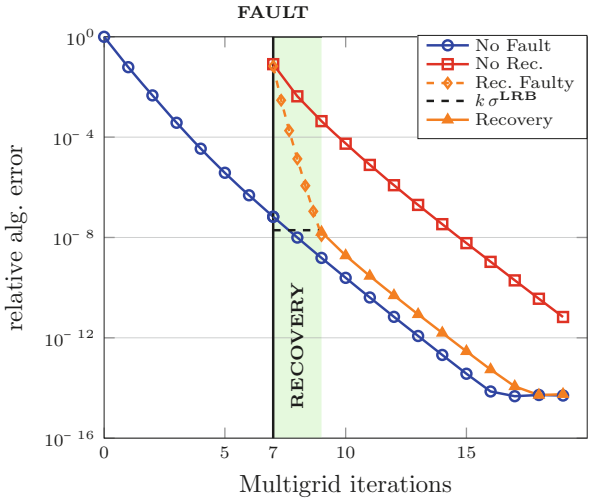


Fig. 11 Comparison of the estimated error using the hierarchically weighted error estimator to the error in a fault-free and a recovery-free scenario. $\kappa\sigma^{LRB}$ denotes the local recoupling threshold for details see [56]

applications. It guards the algorithm against over- or under-solving the subproblem and therefore wasting computational resources.

As already suggested in [54], a so-called *superman strategy* is employed, which increases the computing power locally in the faulty subdomain. The recovery computation is accelerated e.g. through a further subdivision of Ω_F by means of additional shared memory parallelism. In Fig. 11 the development of the estimated error of a recovered solution is compared to the error of a fault-free scenario and a no-recovery scenario. Locally, the number of processes is increased by a factor of 4.

We have also conducted studies with *asynchronous* recovery strategies. Here, the local problem on Ω_F and the problem on the healthy domain $\Omega \setminus \overline{\Omega}_F$ are solved

www.dbooks.org

concurrently. This means that during the recovery process, the multigrid iteration is also continued in parallel in the healthy domain. Different strategies have been explored in [54] which boundary conditions are suitable for the healthy domain on the interface. After the stopping criterion in the faulty domain is fulfilled, the local recovery iteration is terminated and a *recoupling* procedure is initiated.

To emphasize the applicability to large-scale scenarios, the adaptive approach is scaled in [56] to a simulation with about 6.9×10^{11} DoFs and 245, 766 processes run on the JUQUEEN supercomputer. Additionally, the generality of the approach was demonstrated for a scenario where multiple failures in different regions of the domain have been triggered.

3.8 General Performance Issues

Simulations in geophysics require a high spatial resolution which leads to problems with many degrees of freedom. The success of a geophysics simulation framework will thus depend on its scalability and that it has only minimal memory overhead. In the previous sections we already presented scalability results to highlight the results obtained in the TERRANEO project.

However, message passing scalability is not the only metric that is important in high-performance computing. Even more important is the absolute performance which depends critically also on the performance on a single core or a single node. In the extreme scale computing community the relevance of a systematic node-level performance engineering is increasingly being realized, see e.g. [4, 45, 48, 60, 67]. Note that traditional scalability is easier to achieve when the node-level performance is lower. Publishing only scalability results without absolute performance measures is a frequently observed parallel computing cheat, as exposed in [2]. We emphasize here that the TERRANEO project has in this sense profited from long term research efforts in HPC performance engineering. These stem originally from the Dime project[4] [65, 97] and have led to the excellent performance features of the HHG library [12, 13]. Thus, node level performance in TERRANEO is not coming as an afterthought imposed on existing codes, but has been an a priori design goal with high priority.

Within TERRANEO, these techniques were continuously employed to analyze the performance and were used to guide the development of new matrix-free methods. This includes rather simple analysis like calculating the update cost for a single stencil update in terms of floating point operations per second which allows comparing the actual performance to the maximally achievable performance on a given hardware. This type of analysis has been performed in [10] and [11].

In [9] this metric was also extended with an analysis of the memory traffic that is required to apply the stencil to a function which, in terms of linear algebra,

[4] http://dime.fau.de.

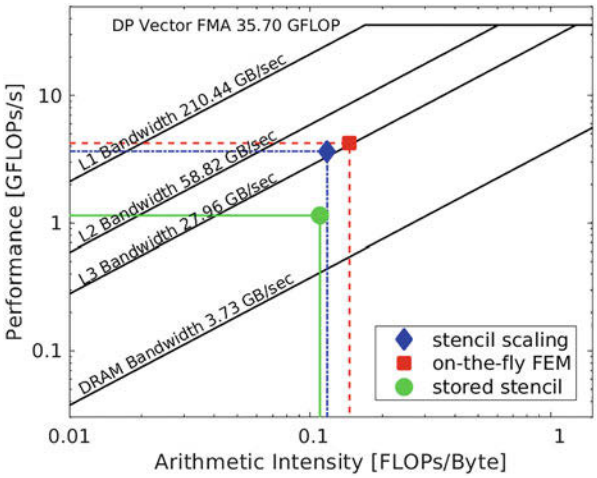


Fig. 12 Roofline model for different approaches presented in [9]

corresponds to matrix-vector multiplication. The analysis provided a lower and upper bound for the performance with respect to memory. One for the optimistic assumption that all data required resides in the last level cache, and one for the pessimistic assumption that everything needs to be loaded from main memory. To evaluate this analysis the Intel Advisor[5] was used as an automatic tool to perform the well established Roofline analysis [58, 103]. The results are given in Fig. 12.

In cooperation with the HPC group of the RRZE in Erlangen, a more sophisticated analysis was performed in [8, 40, 41] where the Execution-Cache-Memory (ECM) model [48] was used to examine the performance.

The Roofline model assumes that only one data location is the bottleneck. This can be the main memory or any of the cache levels represented by the four different limits in Fig. 12. The ECM model however, takes into account memory, all levels of cache, and finally the registers within the CPU. The model helped to design the new computational kernels by identifying performance bottlenecks and guiding subsequent performance optimization steps. The end result of this development procedure are kernels where the analytically predicted optimal duration of a stencil-based update is in good agreement with the measurements in the real program code. Here an error of only $\approx$15% is considered acceptable. In Fig. 13 the occupancy of the execution units within the CPU is shown, as it was derived from the performance models.

[5] https://software.intel.com/advisor.

www.dbooks.org

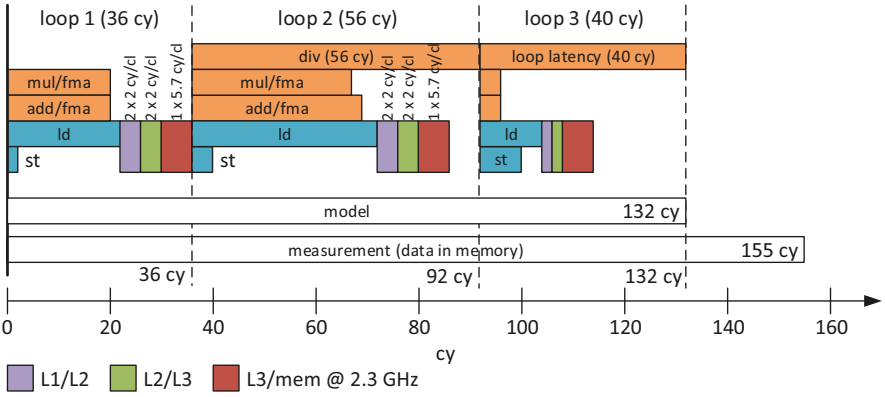


Fig. 13 Duration of the different phases involved during eight stencil-based updates as obtained by the ECM model for the newly developed kernel in [8]

3.9 HyTeG

We successfully demonstrated that the combination of a fully unstructured grid with regular refinement and matrix-free algorithms can be used to solve large geodynamical problems. A substantial amount of work in the TERRANEO project was performed using the HHG prototype. Even though the latter shows excellent performance and scalability, certain issues need to be faced. Due to the fact that the codebase was created over a decade ago, the coding standard cannot live up to current requirements. This makes it hard to maintain the framework and to familiarise new users with it, which is essential for a community code. Furthermore, the prototype was never meant to be used for higher-order discretizations or other than nodal elements.

Therefore a redesign is initiated under a new name: Hybrid Tetrahedral Grids (HYTEG) [63]. In addition to building on the knowledge gathered over the years from developing the HHG framework we could also utilize ideas and infrastructure from the WALBERLA framework [44]. One of the fundamental changes in HYTEG is the strict separation of the data structures that define the macro mesh geometry and the actual simulation data. As in HHG, the tetrahedra of the unstructured macro mesh are split into their geometric primitives, namely volumes, faces, edges and vertices. The lower dimensional primitives (faces, edges and vertices) are used to decouple the volume primitives in terms of communication. Contrary to HHG the parallel partitioning is not solely based on volume primitives, instead all primitives get partitioned between processes. This e.g. allows to take also the computational and memory footprint for face primitives into account for load balancing, for which we can employ sophisticated tools such as ParMetis [61]. Note that in HYTEG the partitioning does not involve global data structures, an essential aspect when entering the exascale era. Like with HHG, it was decided to use C++ as the primary programming language due to its high performance and spread in the HPC

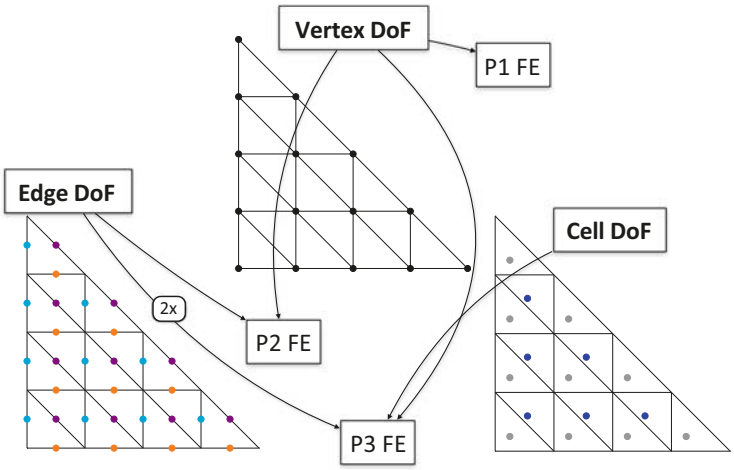


Fig. 14 Basic types can be combined to create various finite element discretizations

community. Up to this point, no compute data are associated with the primitives. In a second step, these can be attached to the primitives. This might be a single value for only some of the primitives or a full hierarchy of grids for every single primitive. This separation also allows for efficient light-weight static and dynamic load balancing techniques similar to those described in [32, 96].

As mentioned above, one major goal when moving from HHG to HYTEG is the realization of different discretizations on hierarchical hybrid grids. In addition to unknowns located at the vertices of each micro-element of the mesh, unknowns on the edges, faces and inside the elements are also supported. These basic types of unknowns are implemented in such a way that different discretizations can be realized by combination which is illustrated in Fig. 14. Using this technique, a Taylor-Hood discretization for a Stokes flow simulation can be realized as well as finite volume discretizations to simulate energy transport [63].

By introducing different types of discretizations, there are also many more compute-intensive kernels that need to be taken care of. The solution chosen in HYTEG to solve this problem is code generation. Inspired by the work in ExaStencils [76, 94], our joint collaboration in this direction [15], and also using experience from pystencils[6] this can be efficiently realized. One difference to other projects that use whole program generation, however, is that only compute intensive kernels are generated, but not the surrounding data structures. In contrast to HYTEG itself, pystencils is using Python, which allows for much more flexibility and metaprogramming capabilities. The generated kernels, however are in C++, which eliminates the need for a Python environment when running HYTEG on supercomputers.

[6] https://i10git.cs.fau.de/pycodegen/pystencils.

www.dbooks.org

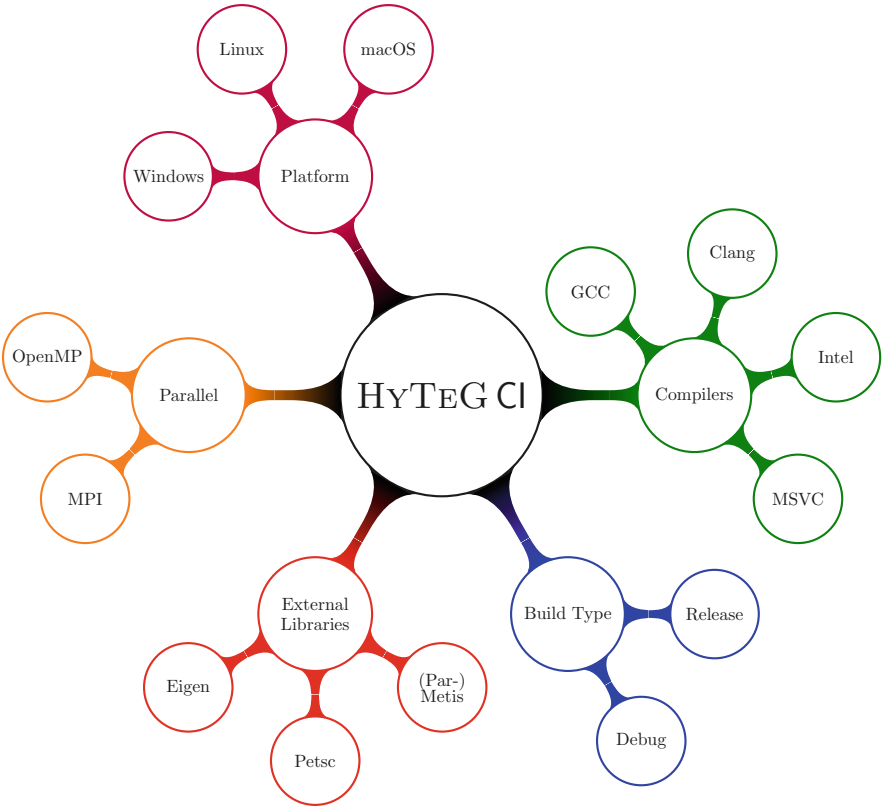


Fig. 15 Each combination of one leaf of every color represents one possible configuration

Another important feature that was introduced with the community code in mind is a state-of-the-art continuous integration process. With every change in the codebase, an automated process is started, which ensures the compatibility with different hardware and software configurations. Additionally, extensive testing is also part of the pipeline to guarantee the correctness of all features inside HyTeG. To illustrate the complexity of this process Fig. 15 shows all possible combinations of configurations.

3.10 Asthenosphere Velocities and Dynamic Topography

In order to demonstrate the flexibility of the different algorithmic components, we have studied selected application questions. In [102] we investigated the question of flow speeds in the asthenosphere. The latter is a mechanically weak layer in the upper mantle right below the lithosphere, the rigid outermost shell of our planet. It

plays an important role in convection modelling as it distinguishes itself from the lower parts of the mantle by a significantly lower viscosity. The precise details of the contrast are, however, unknown. A variety of geologic observations and geodynamic arguments indicates that velocities in the upper mantle may exceed those of tectonic plates by an order of magnitude [49, 51]. The framework was used to simulate high-resolution whole earth convection models with asthenospheric channels of varying thickness. Reduction of the asthenospheric thickness is balanced by an associated reduction in its viscosity, following the *Cathles* parameter [88]. This resulted for the tested end-member case in a set-up with an asthenosphere channel of only 100 km depth and a significant viscosity contrast of up to 4 orders of magnitude relative to the deeper mantle. We found a velocity increase by a factor of 10 between a reference case with an upper mantle of 1000 km depth and the very thin channel end-member case, translating into speeds of ≈ 20 cm/a within the narrow asthenosphere. Our suggested and numerically verified Poiseuille flow model predicts that the upper mantle velocity scales with the inverse of the asthenospheric thickness.

Note that the prototype implemented in HHG already allows to include real-world geophysical data. The model presented in Fig. 16, e.g., uses present-day plate velocities, see [83], as surface boundary conditions and a buoyancy term based on present day temperature and density fields, converted from a seismic tomography model via techniques described in [27]. Viscosity depends on temperature and includes a jump at the bottom of the asthenosphere, chosen at 660 km.

Another geodynamic quantity we studied is *dynamic topography*. This term, which plays a crucial role in our understanding of long term sea-level variations, refers to deflections of the Earth's surface as a response to viscous stresses in the mantle, and has been known from geodynamic arguments for a long time [47, 87]. The precise magnitude of dynamic topography is still debated, however, due to uncertainties in measuring it [18]. Dynamic topography occurs both at the surface

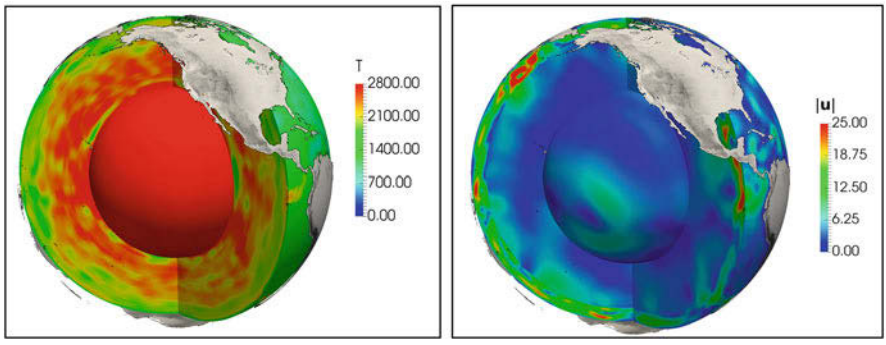


Fig. 16 Mantle convection model computed in HHG with real-world geophysical data: temperature field from seismic tomography for viscosity (left) and resulting flow speeds (right); quantities are non-dimensional

www.dbooks.org

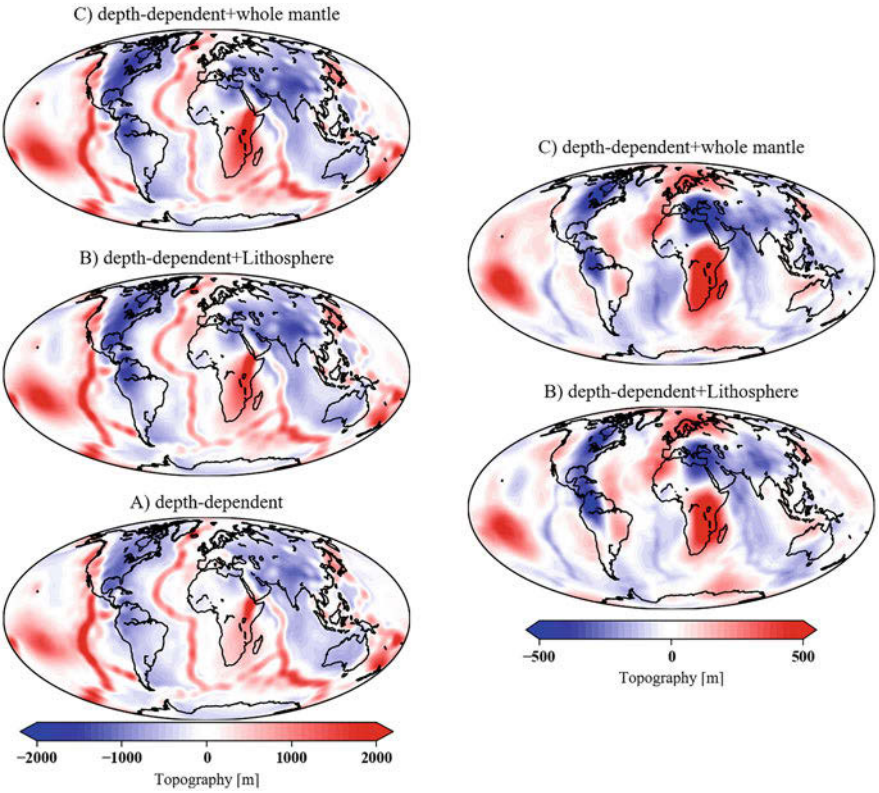


Fig. 17 Surface dynamic topography for cases: A—viscosity with only radial variations, B—with the addition of viscosity variations due to varying thickness of the lithosphere, C—with additional temperature-dependent viscosity in the lower mantle (left) and pairwise difference between scenarios (right)

and the core-mantle boundary (CMB), see [26] for a review. In [10] we studied the influence of lateral viscosity variations on the amplitude and pattern of dynamic topography. We first considered a standard benchmark with a purely radially varying viscosity profile, see [23, 105], for which a semi-analytic solution exists in terms of the propagator matrix technique [47, 85, 87]. This allowed us to verify suitable correctness of our LSQP$_e$ approach from Sect. 3.6. We then compared models with increasing levels of complexity, see Fig. 17, going up to a profile with lateral viscosity variations due to varying thickness of the lithosphere and temperature-dependent variations in the lower mantle. Full details and a discussion of the outcomes can be found in [10].

4 Conclusions and Future Work

TERRANEO is a project in *Computational Science and Engineering* [91]. We successfully developed innovative scientific computing methods for the exascale era. To this end, new algorithms were devised for simulating Earth Mantle convection. We analyzed the accuracy of these methods and their cost on massively parallel computers, showing that our methods exhibit excellent performance. A highly optimized prototype implementation was designed using the HHG multigrid library. Based on HHG, we demonstrated computations with an unprecedented resolution of the Earth's Mantle based on real-world geophyical input data.

In TERRANEO many new methods for future exascale geodynamic simulations were invented: they include new solver components for monolithic multigrid methods for saddle point problems, specially designed smoothers and new strategies to solve the coarse grid problems. Two new classes of matrix-free methods were introduced and analyzed, one based on surrogate polynomials, the other one based on stencil scaling. New scheduling algorithms for parallel multilevel Monte Carlo methods and uncertainty quantification were studied. Methods for fault tolerance were investigated. This includes methods based on in-memory checkpointing as well as new methods for fast algorithmic reconstruction of lost data when hard faults occur. Inverse problems in mantle convection were studied using adjoint techniques.

Careful parallel performance studies complement our work and assess the suitability of the algorithmic components on future exascale computers. Many of the methods were tested on application oriented problems, such as for example a geophysics study on the relation of the thickness of the asthenosphere and upper mantle velocities and the influence of viscosity variations on dynamic topography.

We found that conventional C++-based implementation techniques lack expressiveness and flexibility for massively parallel and highly optimized parallel codes and that achieving performance portability is a major difficulty. Starting from this insight, we invented new programming techniques based on automatic program generation, learning from neighboring SPPEXA projects such as ExaStencils. These ideas have already been realized in HYTEG and WALBERLA as new and innovative simulation software architectures for multiphysics exascale computing.

Thus, some aspects of our research in TERRANEO are of mathematical nature, others fall into the field of computer science. Our methods have also already been used to perform research in geophysics, the *target discipline* of TERRANEO. However, we point out that the research contribution of TERRANEO falls neither into the intersection of all these fields, nor can our contribution be understood from the viewpoint of mathematics or computer science alone. It is also no project in the geosciences, since its primary goal is not the creation of new geophysical insight. The goal of TERRANEO is the construction and analysis of simulation methods that are the enabling technologies for future exascale research in geodynamics.

This result could not be reached by either of the classical disciplines alone. The synthesis of knowledge and methods from mathematics, computer science, and geophysics becomes more than the sum of its parts. We emphasize additionally

www.dbooks.org

that our research in computational science and engineering is not restricted to the application in geophysics. Many of the innovations developed in TERRANEO can be transferred to other *target disciplines*.

With our new computational methods, we have broken previously existing barriers to computational performance. This is demonstrated for example by our solution of indefinite linear systems that have in excess of 10^{13} degrees of freedom. To our knowledge this constitutes the largest finite element computation published to date, even though the computation was still performed on JUQUEEN, a machine that is now outdated and was already retired.

TERRANEO funding in the last period was reduced from the desired four to only three positions so that the new software design and its development could not be completed as originally proposed. A preliminary version of the TERRANEO software will be made public and will contain essential parts of the promised core functionality, but it still lacks most of the application oriented functionality that would make it fully usable as a Geophysics community code. Efforts will be made to continue the development so that the central research goal of TERRANEO can be reached.

Acknowledgments The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. for funding this project by providing computing time on the GCS Supercomputers: SuperMUC at Leibniz Supercomputing Centre (LRZ), JUQUEEN at Jülich Supercomputing Centre (JSC) and Hazel Hen at High-Performance Computing Center Stuttgart (HLRS). Financial support by the German Research Foundation through the Priority Programme 1648 Software for Exascale Computing (SPPEXA), WO671/11-1 and BU 2012/15-1 is greatly acknowledged. We also would like to thank our former project collaborators: D. Bartuschat, J. Eitzinger, B. Gmeiner, M. Hofmann, L. John, H. Stengel, Chr. Waluga, J. Weismüller, G. Wellein, and M. Wittmann.

References

1. Amestoy, P., Buttar, A., L'Excellent, J.Y., Mary, T.: On the complexity of the block low-rank multifrontal factorization. SIAM J. Sci. Comp. **39**(4), A1710–A1740 (2017)
2. Bailey, D.H.: Misleading performance claims in parallel computations. In: Proceedings of the 46th Annual Design Automation Conference, pp. 528–533. ACM, New York (2009)
3. Baker, A., Klawonn, A., Kolev, T., Lanser, M., Rheinbach, O., Yang, U.: Scalability of classical algebraic multigrid for elasticity to half a million parallel tasks. In: Software for Exascale Computing-SPPEXA 2013–2015, pp. 113–140. Springer, Berlin (2016)
4. Bastian, P., Engwer, C., Fahlke, J., Geveler, M., Göddeke, D., Iliev, O., Ippisch, O., Milk, R., Mohring, J., Müthing, S., Ohlberger, M., Ribbrock, D., Turek, S.: Hardware-based efficiency advances in the EXA-DUNE project. In: Software for Exascale Computing - SPPEXA 2013–2015. Lecture Notes in Computational Science and Engineering. Springer, Berlin (2016)
5. Bastian, P., Müller, E.H., Müthing, S., Piatkowski, M.: Matrix-free multigrid block-preconditioners for higher order discontinuous Galerkin discretisations. J. Comput. Phys. **394**, 417–439 (2019). doi:https://doi.org/10.1016/j.jcp.2019.06.001. http://www.sciencedirect.com/science/article/pii/S0021999119303973
6. Bauer, A., Schaal, K., Springel, V., Chandrashekar, P., Pakmor, R., Klingenberg, C.: Simulating turbulence using the astrophysical discontinuous Galerkin code TENET. In: Software for Exascale Computing-SPPEXA 2013–2015, pp. 381–402. Springer, Berlin (2016)

7. Bauer, S., Bunge, H.P., Drzisga, D., Gmeiner, B., Huber, M., John, L., Mohr, M., Rüde, U., Stengel, H., Waluga, C., et al.: Hybrid parallel multigrid methods for geodynamical simulations. In: Software for Exascale Computing-SPPEXA 2013–2015, pp. 211–235. Springer, Berlin (2016). https://doi.org/10.1007/978-3-319-40528-5_10
8. Bauer, S., Mohr, M., Rüde, U., Weismüller, J., Wittmann, M., Wohlmuth, B.: A two-scale approach for efficient on-the-fly operator assembly in massively parallel high performance multigrid codes. Appl. Numer. Math. **122**, 14–38 (2017). https://doi.org/10.1016/j.apnum.2017.07.006
9. Bauer, S., Drzisga, D., Mohr, M., Rüde, U., Waluga, C., Wohlmuth, B.: A stencil scaling approach for accelerating matrix-free finite element implementations. SIAM J. Sci. Comp. **40**(6), C748–C778 (2018). https://doi.org/10.1137/17M1148384
10. Bauer, S., Huber, M., Mohr, M., Rüde, U., Wohlmuth, B.: A new matrix-free approach for large-scale geodynamic simulations and its performance. In: International Conference on Computational Science, pp. 17–30. Springer, Berlin (2018). https://doi.org/10.1007/978-3-319-93701-4_2
11. Bauer, S., Huber, M., Ghelichkhan, S., Mohr, M., Rüde, U., Wohlmuth, B.: Large-scale simulation of mantle convection based on a new matrix-free approach. J. Comp. Sci. **31**, 60–76 (2019). https://doi.org/10.1016/j.jocs.2018.12.006
12. Bergen, B., Hülsemann, F., Rüde, U.: Is 1.7×10^{10} unknowns the largest finite element system that can be solved today? In: SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, pp. 5–5. IEEE, Piscataway (2005)
13. Bergen, B., Gradl, T., Hülsemann, F., Rüde, U.: A massively parallel multigrid method for finite elements. Comp. Sci. Eng. **8**(6), 56–62 (2006)
14. Bielak, J., Ghattas, O., Kim, E.J.: Parallel octree-based finite element method for large-scale earthquake ground motion simulation. Comput. Model. Eng. Sci. **10**(2), 99–112 (2005). https://doi.org/10.3970/cmes.2005.010.099
15. Bolten, M., Franchetti, F., Kelly, P., Lengauer, C., Mohr, M.: Algebraic description and automatic generation of multigrid methods in SPIRAL. Concurrency Comput. Pract. Exp. **29**(17) (2017). Special Issue on Advanced Stencil-Code Engineering. https://doi.org/10.1002/cpe.4105
16. Braess, D., Sarazin, R.: An efficient smoother for the Stokes problem. Appl. Numer. Math. **23**(1), 3–19 (1997). Multilevel methods (Oberwolfach, 1995)
17. Brandt, A., Livne, O.E.: Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics, vol. 67. SIAM, Philadelphia (2011)
18. Braun, J.: The many surface expressions of mantle dynamics. Nat. Geosci. **3**, 825–833 (2010). https://doi.org/10.1038/ngeo1020
19. Brezzi, F., Douglas, J.: Stabilized mixed methods for the Stokes problem. Numer. Math. **53**(1–2), 225–235 (1988)
20. Brown, J.: Efficient nonlinear solvers for nodal high-order finite elements in 3D. J. Sci. Comp. **45**(1–3), 48–63 (2010). https://doi.org/10.1007/s10915-010-9396-8
21. Bunge, H.P., Richards, M.A., Baumgardner, J.R.: Effect of depth-dependent viscosity on the planform of mantle convection. Nature **379**(6564), 436–438 (1996). https://doi.org/10.1038/379436a0
22. Bunge, H.P., Hagelberg, C.R., Travis, B.J.: Mantle circulation models with variational data assimilation: inferring past mantle flow and structure from plate motion histories and seismic tomography. Geophys. J. Int. **152**(2), 280–301 (2003). https://doi.org/10.1046/j.1365-246X.2003.01823.x
23. Burstedde, C., Stadler, G., Alisic, L., Wilcox, L.C., Tan, E., Gurnis, M., Ghattas, O.: Large-scale adaptive mantle convection simulation. Geophys. J. Int. **192**(3), 889–906 (2013). https://doi.org/10.1093/gji/ggs070
24. Clevenger, T.C., Heister, T., Kanschat, G., Kronbichler, M.: A flexible, parallel, adaptive geometric multigrid method for FEM. Technical Report (2019). arXiv:1904.03317

www.dbooks.org

25. Colli, L., Bunge, H.P., Schuberth, B.S.A.: On retrodictions of global mantle flow with assimilated surface velocities. Geophys. Res. Lett. **42**(20), 8341i–8348 (2015). https://doi.org/10.1002/2015gl066001
26. Colli, L., Ghelichkhan, S., Bunge, H.P.: On the ratio of dynamic topography and gravity anomalies in a dynamic earth. Geophys. Res. Lett. **43**(6), 2510–2516 (2016). https://doi.org/10.1002/2016gl067929
27. Colli, L., Ghelichkhan, S., Bunge, H.P., Oeser, J.: Retrodictions of Mid Paleogene mantle flow and dynamic topography in the Atlantic region from compressible high resolution adjoint mantle convection models: sensitivity to deep mantle viscosity and tomographic input model. Gondwana Res. **53**, 252–272 (2018). https://doi.org/10.1016/j.gr.2017.04.027
28. Drzisga, D., Gmeiner, B., Rüde, U., Scheichl, R., Wohlmuth, B.: Scheduling massively parallel multigrid for multilevel Monte Carlo methods. SIAM J. Sci. Comp. **39**(5), S873–S897 (2017). https://doi.org/10.1137/16M1083591
29. Drzisga, D., John, L., Rüde, U., Wohlmuth, B., Zulehner, W.: On the analysis of block smoothers for saddle point problems. SIAM J. Mat. Ana. Appl. **39**(2), 932–960 (2018). https://doi.org/10.1137/16M1106304
30. Drzisga, D., Keith, B., Wohlmuth, B.: The surrogate matrix methodology: a priori error estimation (2019). Preprint arXiv:1902.07333
31. Drzisga, D., Rüde, U., Wohlmuth, B.: Stencil scaling for vector-valued PDEs on hybrid grids with applications to generalized Newtonian fluids (2019). Preprint arXiv:1908.08666
32. Eibl, S., Rüde, U.: A systematic comparison of runtime load balancing algorithms for massively parallel rigid particle dynamics. Comput. Phys. Commun. **244**, 76–85 (2019). https://doi.org/10.1016/j.cpc.2019.06.020
33. Elman, H.C., Silvester, D.J., Wathen, A.J.: Finite Elements and Fast Iterative Solvers: With Applications in Incompressible Fluid Dynamics. Oxford University Press, Oxford (2014)
34. Galdi, G.P., Rannacher, R., Robertson, A.M., Turek, S.: Hemodynamical Flows. Delhi Book Store, New Delhi (2008)
35. Ghelichkhan, S., Bunge, H.P.: The compressible adjoint equations in geodynamics: derivation and numerical assessment. GEM Int. J. Geomath. **7**(1), 1–30 (2016). https://doi.org/10.1007/s13137-016-0080-5
36. Ghelichkhan, S., Bunge, H.P.: The adjoint equations for thermochemical compressible mantle convection: derivation and verification by twin experiments. Proc. R. Soc. A Math. Phys. Eng. Sci. **474**(2220), 20180329 (2018). https://doi.org/10.1098/rspa.2018.0329
37. Gmeiner, B.: Design and analysis of hierarchical hybrid multigrid methods for peta-scale systems and beyond. Ph.D. Thesis, Technische Fakultät der Friedrich-Alexander-Universität Erlangen-Nürnberg (2013)
38. Gmeiner, B., Köstler, H., Stürmer, M., Rüde, U.: Parallel multigrid on hierarchical hybrid grids: a performance study on current high performance computing clusters. Concurrency Comput. Pract. Exp. **26**(1), 217–240 (2014)
39. Gmeiner, B., Waluga, C., Wohlmuth, B.: Local mass-corrections for continuous pressure approximations of incompressible flow. SIAM J. Numer. Ana. **52**(6), 2931–2956 (2014). https://doi.org/10.1137/140959675
40. Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., Wohlmuth, B.: Performance and scalability of hierarchical hybrid multigrid solvers for Stokes systems. SIAM J. Sci. Comp. **37**(2), C143–C168 (2015). https://doi.org/10.1137/130941353
41. Gmeiner, B., Rüde, U., Stengel, H., Waluga, C., Wohlmuth, B.: Towards textbook efficiency for parallel multigrid. Numer. Math. Theory, Methods Appl. **8**(1), 22–46 (2015). https://doi.org/10.4208/nmtma.2015.w10si
42. Gmeiner, B., Huber, M., John, L., Rüde, U., Waluga, C., Wohlmuth, B.: Massively parallel large scale Stokes flow simulation. In: NIC Symposium (2016)
43. Gmeiner, B., Huber, M., John, L., Rüde, U., Wohlmuth, B.: A quantitative performance study for Stokes solvers at the extreme scale. J. Comp. Sci. **17**, 509–521 (2016). https://doi.org/10.1016/j.jocs.2016.06.006

44. Godenschwager, C., Schornbaum, F., Bauer, M., Köstler, H., Rüde, U.: A framework for hybrid parallel flow simulations with a trillion cells in complex geometries. In: SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, pp. 1–12. IEEE, Piscataway (2013)
45. Guillet, T., Pakmor, R., Springel, V., Chandrashekar, P., Klingenberg, C.: High-order magnetohydrodynamics for astrophysics with an adaptive mesh refinement discontinuous Galerkin scheme. Mon. Not. R. Astron. Soc. **485**(3), 4209–4246 (2019)
46. Gupta, S., Wohlmuth, B., Helmig, R.: Multi-rate time stepping schemes for hydro-geomechanical model for subsurface methane hydrate reservoirs. Adv. Water Res. **91**, 78–87 (2016). https://doi.org/10.1016/j.advwatres.2016.02.013
47. Hager, B.H., Clayton, R.W., Richards, M.A., Comer, R.P., Dziewonski, A.M.: Lower mantle heterogeneity, dynamic topography and the geoid. Nature **313**(6003), 541–545 (1985). https://doi.org/10.1038/313541a0
48. Hager, G., Treibig, J., Habich, J., Wellein, G.: Exploring performance and power properties of modern multicore chips via simple machine models. Concurrency Comput. Pract. Exp. **28**, 189–210 (2014)
49. Hartley, R.A., Roberts, G.G., White, N.J., Richardson, C.: Transient convective uplift of an ancient buried landscape. Nat. Geosci. **4**(8), 562–565 (2011). https://doi.org/10.1038/ngeo1191
50. Heister, T., Dannberg, J., Gassmöller, R., Bangerth, W.: High accuracy mantle convection simulation through modern numerical methods - II: realistic models and problems. Geophys. J. Int. **210**(2), 833–851 (2017). https://doi.org/10.1093/gji/ggx195
51. Höink, T., Lenardic, A., Richards, M.A.: Depth-dependent viscosity and mantle stress amplification: implications for the role of the asthenosphere in maintaining plate tectonics. Geophys. J. Int. **191**(1), 30–41 (2012). https://doi.org/10.1111/j.1365-246X.2012.05621.x
52. Horbach, A., Bunge, H.P., Oeser, J.: The adjoint method in geodynamics: derivation from a general operator formulation and application to the initial condition problem in a high resolution mantle circulation model. GEM Int. J. Geomath. **5**(2), 163–194 (2014). https://doi.org/10.1007/s13137-014-0061-5
53. Huber, M., John, L., Pustejovska, P., Rüde, U., Waluga, C., Wohlmuth, B.: Solution techniques for the Stokes system: A priori and a posteriori modifications, resilient algorithms. In: Proceedings of the ICIAM, Beijing. Higher Education Press, Beijing (2015)
54. Huber, M., Gmeiner, B., Rüde, U., Wohlmuth, B.: Resilience for massively parallel multigrid solvers. SIAM J. Sci. Comp. **38**(5), S217–S239 (2016). https://doi.org/10.1137/15M1026122
55. Huber, M., Rüde, U., Waluga, C., Wohlmuth, B.: Surface couplings for subdomain-wise isoviscous gradient based Stokes finite element discretizations. J. Sci. Comp. **74**(2), 895–919 (2018)
56. Huber, M., Rüde, U., Wohlmuth, B.: Adaptive control in roll-forward recovery for extreme scale multigrid. Int. J. High Perf. Comp. Appl. pp. 1–21 (2018). https://doi.org/10.1177/1094342018817088
57. Iaffaldano, G., Lambeck, K.: Pacific plate-motion change at the time of the Hawaiian-Emperor bend constrains the viscosity of Earth's asthenosphere. Geophys. Res. Lett. **41**(10), 3398–3406 (2014). https://doi.org/10.1002/2014GL059763
58. Ilic, A., Pratas, F., Sousa, L.: Cache-aware Roofline model: upgrading the loft. IEEE Comp. Arch. Lett. **13**(1), 21–24 (2014). https://doi.org/10.1109/L-CA.2013.6
59. Ismail-Zadeh, A., Schubert, G., Tsepelev, I., Korotkii, A.: Inverse problem of thermal convection: numerical approach and application to mantle plume restoration. Phys. Earth Planet. Int. **145**(1–4), 99–114 (2004). https://doi.org/10.1016/j.pepi.2004.03.006
60. Jumah, N., Kunkel, J., Zängl, G., Yashiro, H., Dubos, T., Meurdesoif, Y.: GGDML: icosahedral models language extensions. J. Comput. Sci. Technol. Updat. **4**, 1–10 (2017). https://doi.org/10.15379/2410-2938.2017.04.01.01
61. Karypis, G., Kumar, V.: A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. J. Paral. Distrib. Comput. **48**(1), 71–95 (1998). https://doi.org/10.1006/jpdc.1997.1403

www.dbooks.org

62. Klawonn, A., Lanser, M., Rheinbach, O.: Toward extremely scalable nonlinear domain decomposition methods for elliptic partial differential equations. SIAM J. Sci. Comput. **37**(6), C667–C696 (2015). https://doi.org/10.1137/140997907
63. Kohl, N., Thönnes, D., Drzisga, D., Bartuschat, D., Rüde, U.: The HyTeG finite-element software framework for scalable multigrid solvers. Int. J. Parallel Emergent Distrib. Syst. **34**, 1–20 (2018). https://doi.org/10.1080/17445760.2018.1506453
64. Kohl, N., Hötzer, J., Schornbaum, F., Bauer, M., Godenschwager, C., Köstler, H., Nestler, B., Rüde, U.: A scalable and extensible checkpointing scheme for massively parallel simulations. Int. J. High Perfor. Comput. Appl. **33**(4), 571–589 (2019)
65. Kowarschik, M., Rüde, U., Weiss, C., Karl, W.: Cache-aware multigrid methods for solving Poisson's equation in two dimensions. Computing **64**(4), 381–399 (2000)
66. Kronbichler, M., Kormann, K.: A generic interface for parallel cell-based finite element operator application. Comput. Fluids **63**, 135–147 (2012). https://doi.org/10.1016/j.compfluid.2012.04.012
67. Kronbichler, M., Kormann, K.: Fast matrix-free evaluation of discontinuous Galerkin finite element operators. ACM Trans. Math. Softw. **45**(3), 29:1–29:40 (2019). https://doi.org/10.1145/3325864
68. Kronbichler, M., Ljungkvist, K.: Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Trans. Parall. Comput. **6**(1) (2019). https://doi.org/10.1145/3322813
69. Kronbichler, M., Wall, W.A.: A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers. SIAM J. Sci. Comp. **40**(5), A3423–A3448 (2018). https://doi.org/10.1137/16M110455X
70. Kuckuk, S., Köstler, H.: Automatic generation of massively parallel codes from ExaSlang. Computation **4**(3), 27:1–27:20 (2016). Special Issue on High Performance Computing (HPC) Software Design. https://doi.org/10.3390/computation4030027
71. Kunkel, J., Novikova, A., Betke, E.: Towards decoupling the selection of compression algorithms from quality constraints - an investigation of lossy compression efficiency. Supercomput. Front. Innov. **4**, 17–33 (2017). https://doi.org/10.14529/jsfi170402
72. Lambeck, K., Smither, C., Johnston, P.: Sea-level change, glacial rebound and mantle viscosity for northern Europe. Geophys. J. Int. **134**(1), 102–144 (1998). https://doi.org/10.1046/j.1365-246x.1998.00541.x
73. Larin, M., Reusken, A.: A comparative study of efficient iterative solvers for generalized Stokes equations. Numer. Linear Algebra Appl. **15**(1), 13–34 (2008). https://doi.org/10.1002/nla.561
74. Leitenmaier, L.: Data compression for simulation data from Earth mantle convection. Lehrstuhl für Informatik 10 (Systemsimulation), Friedrich-Alexander-Universität Erlangen-Nürnberg. Bachelor Thesis (2014)
75. Lengauer, C., Apel, S., Bolten, M., Chiba, S., Rüde, U., Teich, J., Größlinger, A., Hannig, F., Köstler, H., Claus, L., Grebhahn, A., Groth, S., Kronawitter, S., Kuckuk, S., Rittich, H., Schmitt, C., Schmitt, J.: ExaStencils: Advanced Multigrid Solver Generation (In this volume)
76. Lengauer, C., Apel, S., Bolten, M., Größlinger, A., Hannig, F., Köstler, H., Rüde, U., Teich, J., Grebhahn, A., Kronawitter, S., Kuckuk, S., Rittich, H., Schmitt, C.: ExaStencils: Advanced stencil-code engineering. In: Euro-Par 2014: Parallel Processing Workshops. Lecture Notes in Computer Science, vol. 8806, pp. 553–564. Springer, Berlin (2014). https://doi.org/10.1007/978-3-n-2_47
77. Maitre, J.F., Musy, F., Nigon, P.: A fast solver for the Stokes equations using multigrid with a Uzawa smoother. In: Advances in multigrid methods (Oberwolfach, 1984). Notes on Numerical Fluid Mechanics, vol. 11, pp. 77–83. Braunschweig, Vieweg (1985)
78. May, D.A., Brown, J., Pourhiet, L.L.: A scalable, matrix-free multigrid preconditioner for finite element discretizations of heterogeneous Stokes flow. Comp. Meth. Appl. Mech. Engg. **290**, 496–523 (2015). https://doi.org/10.1016/j.cma.2015.03.014

79. May, D., Sanan, P., Rupp, K., Knepley, M., Smith, B.: Extrem scale multigrid components within PETSc. In: Proceedings of the Platform for Advanced Scientific Computing Conference, PASC (2016)
80. Mitrovica, J.X.: Haskell [1935] revisited. J. Geophys. Res. **101**(B1), 555 (1996). https://doi.org/10.1029/95JB03208
81. Mitrovica, J.X., Forte, A.M.: A new inference of mantle viscosity based upon joint inversion of convection and glacial isostatic adjustment data. Earth Planet. Sci. Lett. **225**(1–2), 177–189 (2004). https://doi.org/10.1016/j.epsl.2004.06.005
82. Mitrovica, J.X., Wahr, J.: Ice age earth rotation. Ann. Rev. Earth Planet. Sci. **39**, 577–616 (2011). https://doi.org/10.1146/annurev-earth-040610-133404
83. Müller, R.D., Sdrolias, M., Gaina, C., Roest, W.R.: Age, spreading rates, and spreading asymmetry of the world's ocean crust. Geochem. Geophys. Geosyst. **9**(4), 1525–2027 (2008)
84. Price, M.G., Davies, J.H.: Profiling the robustness, efficiency and limits of the forward-adjoint method for 3-D mantle convection modelling. Geophys. J. Int. **212**(2), 1450–1462 (2017). https://doi.org/10.1093/gji/ggx489
85. Ricard, Y., Wuming, B.: Inferring the viscosity and the 3-D density structure of the mantle from geoid, topography and plate velocities. Geophys. J. Int. **105**(3), 561–571 (1991). https://doi.org/10.1111/j.1365-246X.1991.tb00796.x
86. Ricard, Y., Spada, G., Sabadini, R.: Polar wandering of a dynamic earth. Geophys. J. Int. **113**(2), 284–298 (1993). https://doi.org/10.1111/j.1365-246X.1993.tb00888.x
87. Richards, M.A., Hager, B.H.: Geoid anomalies in a dynamic earth. J. Geophys. Res. **89**(B7), 5987–6002 (1984). https://doi.org/10.1029/JB089iB07p05987
88. Richards, M.A., Lenardic, A.: The Cathles parameter (Ct): a geodynamic definition of the asthenosphere and implications for the nature of plate tectonics. Geochem. Geophys. Geosyst. **19**(12), 4858–4875 (2018). https://doi.org/10.1029/2018GC007664
89. Richards, M., Bunge, H., Ricard, Y., Baumgardner, J.: Polar wandering in mantle convection models. Geophys. Res. Lett. **26**(12), 1777–1780 (1999). https://doi.org/10.1029/1999GL900331
90. Rüde, U.: Mehrgittermethode—Grundlage der computergestützten Wissenschaften. Informatik Spektrum **42**(2), 138–143 (2019)
91. Rüde, U., Willcox, K., McInnes, L.C., Sterck, H.D.: Research and education in computational science and engineering. SIAM Rev. **60**(3), 707–754 (2018). https://doi.org/10.1137/16M1096840
92. Rudi, J., Malossi, A.C.I., Isaac, T., Stadler, G., Gurnis, M., Staar, P.W.J., Ineichen, Y., Bekas, C., Curioni, A., Ghattas, O.: An extreme-scale implicit solver for complex PDEs: Highly heterogeneous flow in Earth's Mantle. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '15, pp. 5:1–5:12. ACM, New York (2015). https://doi.org/10.1145/2807591.2807675
93. Rudolph, M.L., Lekić, V., Lithgow-Bertelloni, C.: Viscosity jump in Earth's mid-mantle. Science **350**(6266), 1349–1352 (2015). https://doi.org/10.1126/science.aad1929
94. Schmitt, C., Kuckuk, S., Hannig, F., Köstler, H., Teich, J.: ExaSlang: A domain-specific language for highly scalable multigrid solvers. In: Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), pp. 42–51. IEEE Computer Society, Washington (2014)
95. Schöberl, J., Zulehner, W.: On Schwarz-type smoothers for saddle point problems. Numer. Math. **95**(2), 377–399 (2003). https://doi.org/10.1007/s00211-002-0448-3
96. Schornbaum, F., Rüde, U.: Extreme-scale block-structured adaptive mesh refinement. SIAM J. Sci. Comp. **40**(3), C358–C387 (2018)
97. Stals, L., Rüde, U., Weiß, C., Hellwagner, H.: Data local iterative methods for the efficient solution of partial differential equations. In: John, N., Andrew, G., Michael, T. (eds.) Computational Techniques And Applications: Ctac 97-Proceedings Of The Eight Biennial Conference. World Scientific, Singapore (1998)
98. Vanka, S.: Block-implicit multigrid solution of Navier-Stokes equations in primitive variables. J. Comput. Phys. **65**, 138–158 (1986). https://doi.org/10.1016/0021-9991(86)90008-2

www.dbooks.org

99. Verfürth, R.: A multilevel algorithm for mixed problems. SIAM J. Numer. Anal. **21**, 264–271 (1984). https://doi.org/10.1137/0721019
100. Vynnytska, L., Bunge, H.: Restoring past mantle convection structure through fluid dynamic inverse theory: regularisation through surface velocity boundary conditions. GEM - Int. J. Geomath. **6**(1), 83–100 (2014). https://doi.org/10.1007/s13137-014-0060-6
101. Waluga, C., Wohlmuth, B., Rüde, U.: Mass-corrections for the conservative coupling of flow and transport on collocated meshes. J. Comp. Phys. **305**, 319–332 (2016). https://doi.org/10.1016/j.jcp.2015.10.044
102. Weismüller, J., Gmeiner, B., Ghelichkhan, S., Huber, M., John, L., Wohlmuth, B., Rüde, U., Bunge, H.P.: Fast asthenosphere motion in high-resolution global mantle flow models. Geophys. Res. Lett. **42**(18), 7429–7435 (2015). https://doi.org/10.1002/2015GL063727
103. Williams, S., Waterman, A., Patterson, D.: Roofline: An insightful visual performance model for multicore architectures. Commun. ACM **52**(4), 65–76 (2009). https://doi.org/10.1145/1498765.1498785
104. Wobker, H., Turek, S.: Numerical studies of Vanka-type smoothers in computational solid mechanics. Adv. Appl. Math. Mech. **1**(1), 29–55 (2009)
105. Zhong, S., McNamara, A., Tan, E., Moresi, L., Gurnis, M.: A benchmark study on mantle convection in a 3-D spherical shell using CitcomS. Geochem. Geophys. Geosyst. **9**, Q10017 (2008). https://doi.org/10.1029/2008GC002048
106. Zulehner, W.: A class of smoothers for saddle point problems. Computing **65**, 227–246 (2000). https://doi.org/10.1007/s006070070008
107. Zulehner, W.: Analysis of iterative methods for saddle point problems: a unified approach. Math. Comp. **71**(238), 479–505 (2002)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.



Editorial Policy

1. Volumes in the following three categories will be published in LNCSE:

i) Research monographs
ii) Tutorials
iii) Conference proceedings

Those considering a book which might be suitable for the series are strongly advised to contact the publisher or the series editors at an early stage.

2. Categories i) and ii). Tutorials are lecture notes typically arising via summer schools or similar events, which are used to teach graduate students. These categories will be emphasized by Lecture Notes in Computational Science and Engineering. **Submissions by interdisciplinary teams of authors are encouraged.** The goal is to report new developments – quickly, informally, and in a way that will make them accessible to non-specialists. In the evaluation of submissions timeliness of the work is an important criterion. Texts should be well-rounded, well-written and reasonably self-contained. In most cases the work will contain results of others as well as those of the author(s). In each case the author(s) should provide sufficient motivation, examples, and applications. In this respect, Ph.D. theses will usually be deemed unsuitable for the Lecture Notes series. Proposals for volumes in these categories should be submitted either to one of the series editors or to Springer-Verlag, Heidelberg, and will be refereed. A provisional judgement on the acceptability of a project can be based on partial information about the work: a detailed outline describing the contents of each chapter, the estimated length, a bibliography, and one or two sample chapters – or a first draft. A final decision whether to accept will rest on an evaluation of the completed work which should include

– at least 100 pages of text;
– a table of contents;
– an informative introduction perhaps with some historical remarks which should be accessible to readers unfamiliar with the topic treated;
– a subject index.

3. Category iii). Conference proceedings will be considered for publication provided that they are both of exceptional interest and devoted to a single topic. One (or more) expert participants will act as the scientific editor(s) of the volume. They select the papers which are suitable for inclusion and have them individually refereed as for a journal. Papers not closely related to the central topic are to be excluded. Organizers should contact the Editor for CSE at Springer at the planning stage, see *Addresses* below.

In exceptional cases some other multi-author-volumes may be considered in this category.

4. Only works in English will be considered. For evaluation purposes, manuscripts may be submitted in print or electronic form, in the latter case, preferably as pdf- or zipped ps-files. Authors are requested to use the LaTeX style files available from Springer at http://www.springer.com/gp/authors-editors/book-authors-editors/manuscript-preparation/5636 (Click on LaTeX Template → monographs or contributed books).

For categories ii) and iii) we strongly recommend that all contributions in a volume be written in the same LaTeX version, preferably LaTeX2e. Electronic material can be included if appropriate. Please contact the publisher.

Careful preparation of the manuscripts will help keep production time short besides ensuring satisfactory appearance of the finished book in print and online.

www.dbooks.org

5. The following terms and conditions hold. Categories i), ii) and iii):

Authors receive 50 free copies of their book. No royalty is paid.
Volume editors receive a total of 50 free copies of their volume to be shared with authors, but no royalties.

Authors and volume editors are entitled to a discount of 40 % on the price of Springer books purchased for their personal use, if ordering directly from Springer.

6. Springer secures the copyright for each volume.

Addresses:

Timothy J. Barth
NASA Ames Research Center
NAS Division
Moffett Field, CA 94035, USA
barth@nas.nasa.gov

Michael Griebel
Institut für Numerische Simulation
der Universität Bonn
Wegelerstr. 6
53115 Bonn, Germany
griebel@ins.uni-bonn.de

David E. Keyes
Mathematical and Computer Sciences
and Engineering
King Abdullah University of Science
and Technology
P.O. Box 55455
Jeddah 21534, Saudi Arabia
david.keyes@kaust.edu.sa

and

Department of Applied Physics
and Applied Mathematics
Columbia University
500 W. 120 th Street
New York, NY 10027, USA
kd2112@columbia.edu

Risto M. Nieminen
Department of Applied Physics
Aalto University School of Science
and Technology
00076 Aalto, Finland
risto.nieminen@aalto.fi

Dirk Roose
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A
3001 Leuven-Heverlee, Belgium
dirk.roose@cs.kuleuven.be

Tamar Schlick
Department of Chemistry
and Courant Institute
of Mathematical Sciences
New York University
251 Mercer Street
New York, NY 10012, USA
schlick@nyu.edu

Editor for Computational Science
and Engineering at Springer:

Martin Peters
Springer-Verlag
Mathematics Editorial IV
Tiergartenstrasse 17
69121 Heidelberg, Germany
martin.peters@springer.com

Lecture Notes in Computational Science and Engineering

1. D. Funaro, *Spectral Elements for Transport-Dominated Equations.*
2. H.P. Langtangen, *Computational Partial Differential Equations.* Numerical Methods and Diffpack Programming.
3. W. Hackbusch, G. Wittum (eds.), *Multigrid Methods V.*
4. P. Deuflhard, J. Hermans, B. Leimkuhler, A.E. Mark, S. Reich, R.D. Skeel (eds.), *Computational Molecular Dynamics: Challenges, Methods, Ideas.*
5. D. Kröner, M. Ohlberger, C. Rohde (eds.), *An Introduction to Recent Developments in Theory and Numerics for Conservation Laws.*
6. S. Turek, *Efficient Solvers for Incompressible Flow Problems.* An Algorithmic and Computational Approach.
7. R. von Schwerin, ***M**ulti **B**ody **S**ystem **SIM**ulation.* Numerical Methods, Algorithms, and Software.
8. H.-J. Bungartz, F. Durst, C. Zenger (eds.), *High Performance Scientific and Engineering Computing.*
9. T.J. Barth, H. Deconinck (eds.), *High-Order Methods for Computational Physics.*
10. H.P. Langtangen, A.M. Bruaset, E. Quak (eds.), *Advances in Software Tools for Scientific Computing*.
11. B. Cockburn, G.E. Karniadakis, C.-W. Shu (eds.), *Discontinuous Galerkin Methods.* Theory, Computation and Applications.
12. U. van Rienen, *Numerical Methods in Computational Electrodynamics.* Linear Systems in Practical Applications.
13. B. Engquist, L. Johnsson, M. Hammill, F. Short (eds.), *Simulation and Visualization on the Grid.*
14. E. Dick, K. Riemslagh, J. Vierendeels (eds.), *Multigrid Methods VI.*
15. A. Frommer, T. Lippert, B. Medeke, K. Schilling (eds.), *Numerical Challenges in Lattice Quantum Chromodynamics.*
16. J. Lang, *Adaptive Multilevel Solution of Nonlinear Parabolic PDE Systems.* Theory, Algorithm, and Applications.
17. B.I. Wohlmuth, *Discretization Methods and Iterative Solvers Based on Domain Decomposition.*
18. U. van Rienen, M. Günther, D. Hecht (eds.), *Scientific Computing in Electrical Engineering.*
19. I. Babuška, P.G. Ciarlet, T. Miyoshi (eds.), *Mathematical Modeling and Numerical Simulation in Continuum Mechanics.*
20. T.J. Barth, T. Chan, R. Haimes (eds.), *Multiscale and Multiresolution Methods.* Theory and Applications.
21. M. Breuer, F. Durst, C. Zenger (eds.), *High Performance Scientific and Engineering Computing.*
22. K. Urban, *Wavelets in Numerical Simulation.* Problem Adapted Construction and Applications.
23. L.F. Pavarino, A. Toselli (eds.), *Recent Developments in Domain Decomposition Methods.*

www.dbooks.org

24. T. Schlick, H.H. Gan (eds.), *Computational Methods for Macromolecules: Challenges and Applications.*

25. T.J. Barth, H. Deconinck (eds.), *Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics*.

26. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations*.

27. S. Müller, *Adaptive Multiscale Schemes for Conservation Laws*.

28. C. Carstensen, S. Funken, W. Hackbusch, R.H.W. Hoppe, P. Monk (eds.), *Computational Electromagnetics*.

29. M.A. Schweitzer, *A Parallel Multilevel Partition of Unity Method for Elliptic Partial Differential Equations*.

30. T. Biegler, O. Ghattas, M. Heinkenschloss, B. van Bloemen Waanders (eds.), *Large-Scale PDE-Constrained Optimization*.

31. M. Ainsworth, P. Davies, D. Duncan, P. Martin, B. Rynne (eds.), *Topics in Computational Wave Propagation*. Direct and Inverse Problems.

32. H. Emmerich, B. Nestler, M. Schreckenberg (eds.), *Interface and Transport Dynamics.* Computational Modelling.

33. H.P. Langtangen, A. Tveito (eds.), *Advanced Topics in Computational Partial Differential Equations.* Numerical Methods and Diffpack Programming.

34. V. John, *Large Eddy Simulation of Turbulent Incompressible Flows.* Analytical and Numerical Results for a Class of LES Models.

35. E. Bänsch (ed.), *Challenges in Scientific Computing - CISC 2002.*

36. B.N. Khoromskij, G. Wittum, *Numerical Solution of Elliptic Differential Equations by Reduction to the Interface.*

37. A. Iske, *Multiresolution Methods in Scattered Data Modelling.*

38. S.-I. Niculescu, K. Gu (eds.), *Advances in Time-Delay Systems.*

39. S. Attinger, P. Koumoutsakos (eds.), *Multiscale Modelling and Simulation.*

40. R. Kornhuber, R. Hoppe, J. Périaux, O. Pironneau, O. Wildlund, J. Xu (eds.), *Domain Decomposition Methods in Science and Engineering.*

41. T. Plewa, T. Linde, V.G. Weirs (eds.), *Adaptive Mesh Refinement – Theory and Applications.*

42. A. Schmidt, K.G. Siebert, *Design of Adaptive Finite Element Software.* The Finite Element Toolbox ALBERTA.

43. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations II.*

44. B. Engquist, P. Lötstedt, O. Runborg (eds.), *Multiscale Methods in Science and Engineering.*

45. P. Benner, V. Mehrmann, D.C. Sorensen (eds.), *Dimension Reduction of Large-Scale Systems.*

46. D. Kressner, *Numerical Methods for General and Structured Eigenvalue Problems.*

47. A. Boriçi, A. Frommer, B. Joó, A. Kennedy, B. Pendleton (eds.), *QCD and Numerical Analysis III*.

48. F. Graziani (ed.), *Computational Methods in Transport.*

49. B. Leimkuhler, C. Chipot, R. Elber, A. Laaksonen, A. Mark, T. Schlick, C. Schütte, R. Skeel (eds.), *New Algorithms for Macromolecular Simulation.*

50. M. Bücker, G. Corliss, P. Hovland, U. Naumann, B. Norris (eds.), *Automatic Differentiation: Applications, Theory, and Implementations.*

51. A.M. Bruaset, A. Tveito (eds.), *Numerical Solution of Partial Differential Equations on Parallel Computers.*

52. K.H. Hoffmann, A. Meyer (eds.), *Parallel Algorithms and Cluster Computing.*

53. H.-J. Bungartz, M. Schäfer (eds.), *Fluid-Structure Interaction.*

54. J. Behrens, *Adaptive Atmospheric Modeling.*

55. O. Widlund, D. Keyes (eds.), *Domain Decomposition Methods in Science and Engineering XVI.*

56. S. Kassinos, C. Langer, G. Iaccarino, P. Moin (eds.), *Complex Effects in Large Eddy Simulations.*

57. M. Griebel, M.A Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations III.*

58. A.N. Gorban, B. Kégl, D.C. Wunsch, A. Zinovyev (eds.), *Principal Manifolds for Data Visualization and Dimension Reduction.*

59. H. Ammari (ed.), *Modeling and Computations in Electromagnetics: A Volume Dedicated to Jean-Claude Nédélec.*

60. U. Langer, M. Discacciati, D. Keyes, O. Widlund, W. Zulehner (eds.), *Domain Decomposition Methods in Science and Engineering XVII.*

61. T. Mathew, *Domain Decomposition Methods for the Numerical Solution of Partial Differential Equations.*

62. F. Graziani (ed.), *Computational Methods in Transport: Verification and Validation.*

63. M. Bebendorf, *Hierarchical Matrices.* A Means to Efficiently Solve Elliptic Boundary Value Problems.

64. C.H. Bischof, H.M. Bücker, P. Hovland, U. Naumann, J. Utke (eds.), *Advances in Automatic Differentiation.*

65. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations IV*.

66. B. Engquist, P. Lötstedt, O. Runborg (eds.), *Multiscale Modeling and Simulation in Science.*

67. I.H. Tuncer, Ü. Gülcat, D.R. Emerson, K. Matsuno (eds.), *Parallel Computational Fluid Dynamics 2007.*

68. S. Yip, T. Diaz de la Rubia (eds.), *Scientific Modeling and Simulations*.

69. A. Hegarty, N. Kopteva, E. O'Riordan, M. Stynes (eds.), *BAIL* 2008 – *Boundary and Interior Layers.*

70. M. Bercovier, M.J. Gander, R. Kornhuber, O. Widlund (eds.), *Domain Decomposition Methods in Science and Engineering XVIII.*

71. B. Koren, C. Vuik (eds.), *Advanced Computational Methods in Science and Engineering.*

72. M. Peters (ed.), *Computational Fluid Dynamics for Sport Simulation.*

73. H.-J. Bungartz, M. Mehl, M. Schäfer (eds.), *Fluid Structure Interaction II - Modelling, Simulation, Optimization.*

74. D. Tromeur-Dervout, G. Brenner, D.R. Emerson, J. Erhel (eds.), *Parallel Computational Fluid Dynamics 2008.*

75. A.N. Gorban, D. Roose (eds.), *Coping with Complexity: Model Reduction and Data Analysis.*

www.dbooks.org

76. J.S. Hesthaven, E.M. Rønquist (eds.), *Spectral and High Order Methods for Partial Differential Equations.*

77. M. Holtz, *Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance*.

78. Y. Huang, R. Kornhuber, O.Widlund, J. Xu (eds.), *Domain Decomposition Methods in Science and Engineering XIX*.

79. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations V*.

80. P.H. Lauritzen, C. Jablonowski, M.A. Taylor, R.D. Nair (eds.), *Numerical Techniques for Global Atmospheric Models*.

81. C. Clavero, J.L. Gracia, F.J. Lisbona (eds.), *BAIL 2010 – Boundary and Interior Layers, Computational and Asymptotic Methods*.

82. B. Engquist, O. Runborg, Y.R. Tsai (eds.), *Numerical Analysis and Multiscale Computations*.

83. I.G. Graham, T.Y. Hou, O. Lakkis, R. Scheichl (eds.), *Numerical Analysis of Multiscale Problems*.

84. A. Logg, K.-A. Mardal, G. Wells (eds.), *Automated Solution of Differential Equations by the Finite Element Method*.

85. J. Blowey, M. Jensen (eds.), *Frontiers in Numerical Analysis - Durham 2010.*

86. O. Kolditz, U.-J. Gorke, H. Shao, W. Wang (eds.), *Thermo-Hydro-Mechanical-Chemical Processes in Fractured Porous Media - Benchmarks and Examples*.

87. S. Forth, P. Hovland, E. Phipps, J. Utke, A. Walther (eds.), *Recent Advances in Algorithmic Differentiation*.

88. J. Garcke, M. Griebel (eds.), *Sparse Grids and Applications*.

89. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations VI.*

90. C. Pechstein, *Finite and Boundary Element Tearing and Interconnecting Solvers for Multiscale Problems*.

91. R. Bank, M. Holst, O. Widlund, J. Xu (eds.), *Domain Decomposition Methods in Science and Engineering XX*.

92. H. Bijl, D. Lucor, S. Mishra, C. Schwab (eds.), *Uncertainty Quantification in Computational Fluid Dynamics*.

93. M. Bader, H.-J. Bungartz, T. Weinzierl (eds.), *Advanced Computing*.

94. M. Ehrhardt, T. Koprucki (eds.), *Advanced Mathematical Models and Numerical Techniques for Multi-Band Effective Mass Approximations*.

95. M. Azaïez, H. El Fekih, J.S. Hesthaven (eds.), *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2012*.

96. F. Graziani, M.P. Desjarlais, R. Redmer, S.B. Trickey (eds.), *Frontiers and Challenges in Warm Dense Matter*.

97. J. Garcke, D. Pflüger (eds.), *Sparse Grids and Applications – Munich 2012*.

98. J. Erhel, M. Gander, L. Halpern, G. Pichot, T. Sassi, O. Widlund (eds.), *Domain Decomposition Methods in Science and Engineering XXI.*

99. R. Abgrall, H. Beaugendre, P.M. Congedo, C. Dobrzynski, V. Perrier, M. Ricchiuto (eds.), *High Order Nonlinear Numerical Methods for Evolutionary PDEs - HONOM 2013.*

100. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations VII.*

101. R. Hoppe (ed.), *Optimization with PDE Constraints - OPTPDE 2014*.

102. S. Dahlke, W. Dahmen, M. Griebel, W. Hackbusch, K. Ritter, R. Schneider, C. Schwab, H. Yserentant (eds.), *Extraction of Quantifiable Information from Complex Systems*.

103. A. Abdulle, S. Deparis, D. Kressner, F. Nobile, M. Picasso (eds.), *Numerical Mathematics and Advanced Applications - ENUMATH 2013*.

104. T. Dickopf, M.J. Gander, L. Halpern, R. Krause, L.F. Pavarino (eds.), *Domain Decomposition Methods in Science and Engineering XXII*.

105. M. Mehl, M. Bischoff, M. Schäfer (eds.), *Recent Trends in Computational Engineering - CE2014*. Optimization, Uncertainty, Parallel Algorithms, Coupled and Complex Problems.

106. R.M. Kirby, M. Berzins, J.S. Hesthaven (eds.), *Spectral and High Order Methods for Partial Differential Equations - ICOSAHOM'14*.

107. B. Jüttler, B. Simeon (eds.), *Isogeometric Analysis and Applications 2014*.

108. P. Knobloch (ed.), *Boundary and Interior Layers, Computational and Asymptotic Methods – BAIL 2014*.

109. J. Garcke, D. Pflüger (eds.), *Sparse Grids and Applications – Stuttgart 2014*.

110. H. P. Langtangen, *Finite Difference Computing with Exponential Decay Models*.

111. A. Tveito, G.T. Lines, *Computing Characterizations of Drugs for Ion Channels and Receptors Using Markov Models*.

112. B. Karazösen, M. Manguoğlu, M. Tezer-Sezgin, S. Göktepe, Ö. Uğur (eds.), *Numerical Mathematics and Advanced Applications - ENUMATH 2015*.

113. H.-J. Bungartz, P. Neumann, W.E. Nagel (eds.), *Software for Exascale Computing - SPPEXA 2013-2015*.

114. G.R. Barrenechea, F. Brezzi, A. Cangiani, E.H. Georgoulis (eds.), *Building Bridges: Connections and Challenges in Modern Approaches to Numerical Partial Differential Equations*.

115. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differential Equations VIII*.

116. C.-O. Lee, X.-C. Cai, D.E. Keyes, H.H. Kim, A. Klawonn, E.-J. Park, O.B. Widlund (eds.), *Domain Decomposition Methods in Science and Engineering XXIII*.

117. T. Sakurai, S.-L. Zhang, T. Imamura, Y. Yamamoto, Y. Kuramashi, T. Hoshi (eds.), *Eigenvalue Problems: Algorithms, Software and Applications in Petascale Computing*. EPASA 2015, Tsukuba, Japan, September 2015.

118. T. Richter (ed.), *Fluid-structure Interactions*. Models, Analysis and Finite Elements.

119. M.L. Bittencourt, N.A. Dumont, J.S. Hesthaven (eds.), *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2016.* Selected Papers from the ICOSAHOM Conference, June 27-July 1, 2016, Rio de Janeiro, Brazil.

120. Z. Huang, M. Stynes, Z. Zhang (eds.), *Boundary and Interior Layers, Computational and Asymptotic Methods BAIL 2016.*

121. S.P.A. Bordas, E.N. Burman, M.G. Larson, M.A. Olshanskii (eds.), *Geometrically Unfitted Finite Element Methods and Applications*. Proceedings of the UCL Workshop 2016.

www.dbooks.org

122. A. Gerisch, R. Penta, J. Lang (eds.), *Multiscale Models in Mechano and Tumor Biology*. Modeling, Homogenization, and Applications.

123. J. Garcke, D. Pflüger, C.G. Webster, G. Zhang (eds.), *Sparse Grids and Applications - Miami 2016*.

124. M. Schäfer, M. Behr, M. Mehl, B. Wohlmuth (eds.), *Recent Advances in Computational Engineering*. Proceedings of the 4th International Conference on Computational Engineering (ICCE 2017) in Darmstadt.

125. P.E. Bjørstad, S.C. Brenner, L. Halpern, R. Kornhuber, H.H. Kim, T. Rahman, O.B. Widlund (eds.), *Domain Decomposition Methods in Science and Engineering XXIV*. 24th International Conference on Domain Decomposition Methods, Svalbard, Norway, February 6–10, 2017.

126. F.A. Radu, K. Kumar, I. Berre, J.M. Nordbotten, I.S. Pop (eds.), *Numerical Mathematics and Advanced Applications – ENUMATH 2017*.

127. X. Roca, A. Loseille (eds.), *27th International Meshing Roundtable*.

128. Th. Apel, U. Langer, A. Meyer, O. Steinbach (eds.), *Advanced Finite Element Methods with Applications*. Selected Papers from the 30th Chemnitz Finite Element Symposium 2017.

129. M. Griebel, M.A. Schweitzer (eds.), *Meshfree Methods for Partial Differencial Equations IX*.

130. S. Weißer, BEM-based Finite Element *Approaches on Polytopal Meshes*.

131. V.A. Garanzha, L. Kamenski, H. Si (eds.), *Numerical Geometry, Grid Generation and Scientific Computing*. Proceedings of the 9th International Conference, NUMGRID2018/Voronoi 150, Celebrating the 150th Anniversary of G. F. Voronoi, Moscow, Russia, December 2018.

132. H. van Brummelen, A. Corsini, S. Perotto, G. Rozza (eds.), *Numerical Methods for Flows*.

133. ——

134. S.J. Sherwin, D. Moxey, J. Peiro, P.E. Vincent, C. Schwab (eds.), *Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2018*.

135. G.R. Barrenechea, J. Mackenzie (eds.), *Boundary and Interior Layers, Computational and Asymptotic Methods BAIL 2018*.

136. H.-J. Bungartz, S. Reiz, B. Uekermann, P. Neumann, W.E. Nagel (eds.), *Software for Exascale Computing - SPPEXA 2016–2019*.

For further information on these books please have a look at our mathematics catalogue at the following URL: www.springer.com/series/3527

Monographs in Computational Science and Engineering

1. J. Sundnes, G.T. Lines, X. Cai, B.F. Nielsen, K.-A. Mardal, A. Tveito, *Computing the Electrical Activity in the Heart.*

For further information on this book, please have a look at our mathematics catalogue at the following URL: www.springer.com/series/7417

Texts in Computational Science and Engineering

1. H. P. Langtangen, *Computational Partial Differential Equations.* Numerical Methods and Diffpack Programming. 2nd Edition
2. A. Quarteroni, F. Saleri, P. Gervasio, *Scientific Computing with MATLAB and Octave.* 4th Edition
3. H. P. Langtangen, *Python Scripting for Computational Science*. 3rd Edition
4. H. Gardner, G. Manduchi, *Design Patterns for e-Science*.
5. M. Griebel, S. Knapek, G. Zumbusch, *Numerical Simulation in Molecular Dynamics*.
6. H. P. Langtangen, *A Primer on Scientific Programming with Python.* 5th Edition
7. A. Tveito, H. P. Langtangen, B. F. Nielsen, X. Cai, *Elements of Scientific Computing.*
8. B. Gustafsson, *Fundamentals of Scientific Computing.*
9. M. Bader, *Space-Filling Curves.*
10. M. Larson, F. Bengzon, *The Finite Element Method: Theory, Implementation and Applications.*
11. W. Gander, M. Gander, F. Kwok, *Scientific Computing: An Introduction using Maple and MATLAB.*
12. P. Deuflhard, S. Röblitz, *A Guide to Numerical Modelling in Systems Biology.*
13. M. H. Holmes, *Introduction to Scientific Computing and Data Analysis*.
14. S. Linge, H. P. Langtangen, *Programming for Computations* - A Gentle Introduction to Numerical Simulations with MATLAB/Octave.
15. S. Linge, H. P. Langtangen, *Programming for Computations* - A Gentle Introduction to Numerical Simulations with Python.
16. H.P. Langtangen, S. Linge, *Finite Difference Computing with PDEs* - A Modern Software Approach.
17. B. Gustafsson, *Scientific Computing from a Historical Perspective*.
18. J. A. Trangenstein, *Scientific Computing*. Volume I - Linear and Nonlinear Equations.

www.dbooks.org

19. J. A. Trangenstein, *Scientific Computing*. Volume II - Eigenvalues and Optimization.

20. J. A. Trangenstein, *Scientific Computing*. Volume III - Approximation and Integration.

For further information on these books please have a look at our mathematics catalogue at the following URL: www.springer.com/series/5151