THE EXPERT'S VOICE® IN ENTERPRISE COMPUTING AND POWER MANAGEMENT

Energy Efficient Servers

Blueprints for Data Center Optimization

THE IT PROFESSIONAL'S OPERATIONAL HANDBOOK

Corey Gough, Ian Steiner, and Winston A. Saunders

Apress open

www.dbooks.org

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.



Apress®

Contents at a Glance

About the Authors xv

About the Technical Reviewers xvii

Contributing Authors xix

Acknowledgments xxi

■ Chapter 1: Why Data Center Efficiency Matters 1

■ Chapter 2: CPU Power Management 21

■ Chapter 3: Memory and I/O Power Management 71

■ Chapter 4: Platform Power Management 93

■ Chapter 5: BIOS and Management Firmware 153

■ Chapter 6: Operating Systems 173

■ Chapter 7: Monitoring 209

■ Chapter 8: Characterization and Optimization 269

■ Chapter 9: Data Center Management 307

■ Appendix A: Technology and Terms 319

Index 327

www.dbooks.org

CHAPTER 1

Why Data Center Efficiency Matters

Data centers are the information factories that shape our modern experience. When we access online information ranging from reading our personal email and the news to engaging in commerce, using social media, and consuming entertainment, we are depending on data centers, which provide the computational backbone for the Internet. They create many of the movies we watch, design the cars we drive, and optimize the airplanes we fly. They are used to make scientific discoveries, to find oil, and to predict the spread of disease. Data centers are at the heart of the digital economy.

In 2010, about 30 million servers were in operation worldwide,[1] and the number has been increasing annually. The growth of the Internet of Things[2] is expected to increase the number of connected devices to over 25 billion by 2020. Other factors driving growth include the continued "dematerialization" of goods,[3] the growth of the worldwide economy,[4] and the increased expectation that our lives are connected to one another through computing technology.

From the perspective of overall energy use, centralized data center–based computing in modern facilities is highly efficient. Recently Facebook estimated that the energy used to sustain an average account for a month is about equal to the energy used to make a cup of coffee.[5] eBay's published data center energy use[6] shows that the amount of carbon produced per transaction is about 50 times lower than the carbon produced in a short drive to the store to complete the same purchase.[7] One recent study found that

[1]Jonathan G. Koomey, *Growth in Data Center Electricity Use 2005 TO 2010* (Oakland, CA: Analytics Press, 2011), http://analyticspress.com/datacenters.html.
[2]See www.gartner.com/newsroom/id/2636073.
[3]See http://gigaom.com/2010/04/29/greennet-the-dematerialization-opportunity/.
[4]See, for example, John M. Jordan, *Information, Technology and Innovation: Resources for Growth in a Connected World* (New York: Wiley, 2012).
[5]See www.facebook.com/green/app_439663542812831.
[6]See http://tech.ebay.com/dashboard.
[7]See www.datacenterknowledge.com/archives/2013/03/12/why-ebays-digital-service-efficiency-changes-the-game/.

online purchasing of music uses 40%–80% less energy than any of multiple methods for delivering music by CD, even though that calculation used an upper bound estimate for the electricity intensity of Internet data transfers.[8]

It's somewhat ironic that a principal driver of efficiency in data centers, namely scale, also attracts the most attention to the energy use by data centers. Large-scale data centers can share more resources; for instance, in the case of $N + 1$ redundancy of critical infrastructure systems such as air handlers or power back-up systems,[9] the incremental penalty decreases as size, and therefore N, increases. However, because of their scale, data centers also require large amounts of electrical energy to operate. Typical large-scale data centers require tens of megawatts of electrical power—enough power to sustain a small city. It is in part this high localized energy use that attracts attention to data centers—they are large and visible buildings that consume a lot of energy. As a result, they can attract the scrutiny of both social activists,[10] neighbors,[11] and legislators.[12]

An Industry's Call to Action

It was the convergence of two unrelated events that brought attention to data center energy use. The first was the growth in scale of data centers and the Internet. By one estimate, the number of adults logging onto the Internet increased by 37% from 2000 to 2004. The other trend was the growth of computing performance primarily through clock speed and efficiencies increases.[13] The result of both growing numbers of data centers and growing power use by the servers (driven by numbers of servers, only marginally by power use per server) within the data center was explosive growth in the power consumed by the data center. Although overstated, claims of "economic meltdown" of the data center certainly grabbed attention.[14]

In response to rising public awareness of data center energy use, Congress commissioned a 2007 analysis of US data center energy consumption.[15] The work, completed in 2007 by Lawrence Berkeley National Laboratory using a "bottoms-up" methodology, estimated that data centers were consuming about 1.5% of US electrical energy. Even more alarming, by 2006, data center energy use had doubled since the year 2000 and was on track to almost double again over the following five years.

---

[8]Christopher Weber, Jonathan G. Koomey, and Scott Matthews, "The Energy and Climate Change Impacts of Different Music Delivery Methods," *Journal of Industrial Ecology* 14, no. 5 (October 2010): 754–769, http://dx.doi.org/10.1111/j.1530-9290.2010.00269.x.

[9]See www.lifelinedatacenters.com/data-center/ups-configuration-redundancy/.

[10]See www.greenbiz.com/blog/2011/12/15/facebook-ends-greenpeace-campaign-major-green-commitments.

[11]See http://news.idg.no/cw/art.cfm?id=7C75C477-1A64-67EA-E4F528FE768FA524.

[12]See www.whitehouse.gov/blog/2014/09/30/better-buildings-challenge-expands-take-data-centers/.

[13]See Jonathan G. Koomey, Stephen Berard, Marla Sanchez, and Henry Wong, "Implications of Historical Trends in the Electrical Efficiency of Computing," *IEEE Annals of the History of Computing* 33, no. 3 (July–September 2011): 46–54, http://doi.ieeecomputersociety.org/10.1109/MAHC.2010.28.

[14]Ken Brill, "The Economic Meltdown of Moore's Law and the Green Data Center," (2007) www.usenix.org/legacy/event/lisa07/tech/brill_talk.pdf.

[15]See www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study.

www.dbooks.org

The report flagged the concern that without concerted effort within the data center industry to improve efficiency, the growth of energy consumption risked becoming unsupportable with implications not only for the industries directly affected, but for the economy itself.

The report highlighted some opportunities to improve efficiency and painted several achievable scenarios. Among areas identified for improvement with the biggest impact were data center infrastructure efficiency and the IT equipment inside the data centers. Although the efficiency of the IT equipment in data centers, and specifically the servers, is the focus of this book, it is worthwhile to discuss some of the progress that has been made in improving the efficiency of the infrastructure of data centers.

Data Center Infrastructure Energy Use

The infrastructure energy use of data centers, meaning the energy used to provide clean, reliable, uninterrupted power to the IT equipment and also to remove the waste heat generated by the equipment, is an important part of the overall energy use by data centers. In many cases, the infrastructure can consume a substantial portion of the overall energy use of the data center. Figure 1-1 shows the power consumption of a data center, divided into infrastructure (of non-IT power) and the IT equipment power consumption. Since non-IT power does not contribute directly to information processing, it is considered to contribute to the inefficiency of the data center.

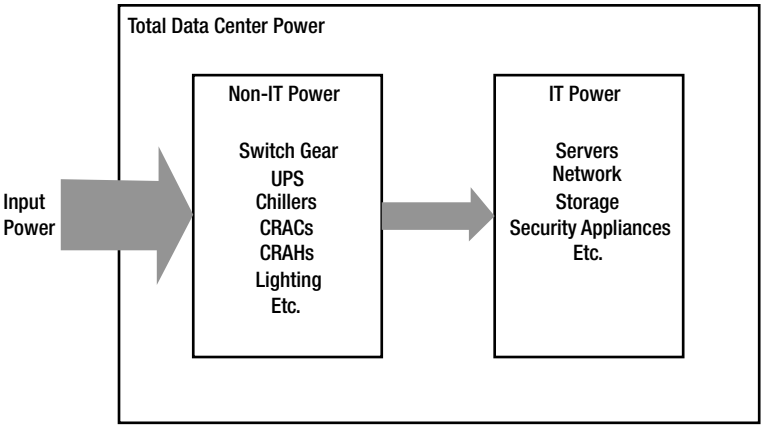


Figure 1-1. *The power consumption of a data center*

Since the infrastructure exists only to provide support to the IT equipment by maintaining acceptable environmental factors and ensuring clean uninterrupted power delivery, it is considered to be an overhead power usage. On the other hand, the IT equipment is contributing directly to the information processing, and hence is directly related to the efficiency of the data center. This is illustrated schematically in Figure 1-1.

The accepted metric for infrastructure efficiency is the power usage effectiveness (PUE), defined as the ratio of the total energy use by the data center to that of the energy used by the IT equipment.

$$PUE = \frac{Total\ Data\ Center\ Energy\ Use}{IT\ Equipment\ Energy\ Use}$$

Typical enterprise data centers that were designed to now outdated computer room building standards typically would have had a PUE in the range of two to three.[16] That means that for one watt of power used to run the computer, one to two watts of power are used to supply power and provide cooling for the IT equipment. By modern standards, this is highly inefficient. Figure 1-2 illustrates the inverse relationship between data center infrastructure and PUE. For PUE = 2.0, 50% of the power in the data center is used for non-computational purposes. Some highly inefficient data centers can operate at a PUE > 3. As PUE increases above 2.0, over 50% of the data center power is used for heating, cooling, and power conditioning.

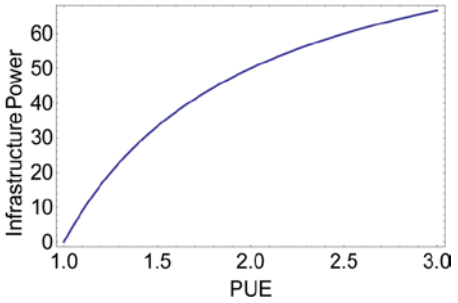


Figure 1-2. *The fraction of total data center power used by data center infrastructure as a function of the PUE*

Through work done by industry groups like the Green Grid,[17] standard methods to improve infrastructure efficiency have been defined and implemented across the industry. These have resulted in dramatic improvements in the PUE values of state-of-the-art data centers.

A commonly discussed potential weakness of PUE as a metric of data center efficiency is that the very inefficiencies PUE addresses, those of moving air for cooling and conditioning electrical power for delivery, also exist within the server (and thus the IT equipment) itself. Although this is true, the incentive to improve the server by optimizing its energy efficiency lies with the system manufacturer (as will be discussed later in this chapter). PUE provides a metric the designer and operator of the data center facility can use to optimize what is within their control. It is for this reason PUE has been such a successful driver of overall data center efficiency.

---

[16]Victor Avelar, Dan Azevedo, Alan French, eds., "PUE: A Comprehensive Examination of the Metric," White Paper #49 (2013), www.thegreengrid.org/~/media/WhitePapers/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf?lang=en

[17]See www.thegreengrid.org/.

www.dbooks.org

Purpose-built mega data centers—like those of Yahoo!, Facebook, and Google—are heavily reliant upon free-air cooling.[18] Typical PUE values in these data centers are about 1.1, meaning of the energy being consumed by the data center, only 10% is being used for non-compute-related tasks. Other, more conventional recently constructed data centers have PUE values near 1.4, meaning about 40% of the energy used by the data center goes to support infrastructure. The reasons these values are higher than the purpose-built mega data centers has to do with specific architectural choices, such as cooling design, as well as requirements for equipment redundancy to meet business-specific resiliency goals.

Although new data center construction typically follows industry best practices for efficient design, improving the efficiency of older, legacy data centers remains a persistent problem. There are several root causes of this. One of these is the rapid evolution of data center technology. For instance, as recently as 2011, ASHRAE approved new building standards that encourage higher operating temperatures in many types of data center.[19] Typically higher operating temperatures have been reported to reduce infrastructure energy use by up to 4% per degree Celsius,[20] a substantial savings.[21]

Data centers have been operated between 68 and 72 F, mostly for historical reasons. Cooling requirements in older IT equipment and mainframe computers were less well understood and placed heavy reliance on room cooling because of their scale and size.[22] A room-sized computer demands room sized cooling. With the migration toward the current generation of servers, the cooling requirements of the servers have changed, but room specifications have been slow to follow.

Although the higher temperature set point can be adjusted in older buildings, air flow management systems may not be designed or optimized to mitigate localized hot spots in the data center. Unless hot spots are carefully managed, this can lead to increased risk for service availability unless the architecture is substantially changed. Since data center buildings are typically depreciated on a 10- to 20-year schedule, it's not entirely surprising that the timescale for the majority of data centers to catch up with current best practices, let alone match future advances, is on the order of years. At this point, much of the technical innovation for improved data center infrastructure is completed or known, and it is simply a matter of time for current practice to catch up with best practices.

Energy Proportional Server Efficiency

Nearly simultaneously with the report to the US Congress on data center energy consumption, an influential paper published by Luiz André Barroso and Urs Hölzle of Google[23] introduced the concept of energy proportional computing. Computing efficiency depends on both the computational work output of the server as well as the energy consumed by the server. The key insight of the energy proportional model was

[18]See www.google.com/green/efficiency/datacenters.
[19]*Thermal Guidelines for Data Processing Environments, 3rd ed.* (ASHRAE, 2012).
[20]See www.datacenterknowledge.com/archives/2007/09/24/data-center-cooling-set-points-debated/.
[21]More careful studies of this savings appear to be warranted.
[22]See www.intel.com/content/www/us/en/data-center-efficiency/efficient-datacenter-high-ambient-temperature-operation-brief.html.
[23]See www.barroso.org/publications/ieee_computer07.pdf.

the realization that bringing server efficiency closer to the theoretical maximum at all workload conditions would improve overall data center efficiency. By ensuring server energy use scaled proportionally to workload, the efficiency of the servers is optimized over a wider range of utilization, as shown in Figure 1-3. The figure on the left shows the power consumption of a server (ca. 2006) whose idle power is 70% of the peak power. Because power consumption does not scale with workload, the efficiency is far below peak at most operating conditions. The figure on the right shows a server with idle power which is 20% of peak. In this case the efficiency is much higher at all utilization points.

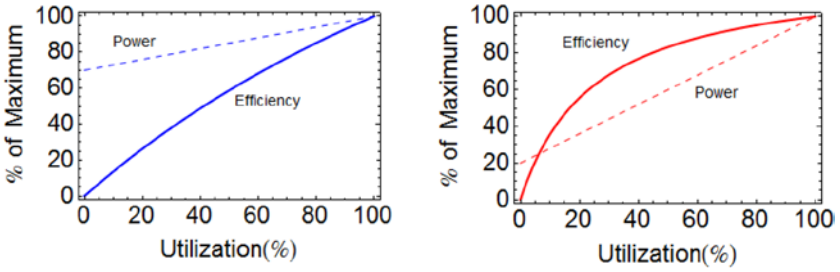


Figure 1-3. *The power consumption and efficiency of two model servers*

Most servers in 2007 consumed almost the same power at 0% utilization (i.e., doing no computations) as they consumed at 100% utilization (i.e., doing the maximum workload or computations per second). For instance, one of the earliest systems reported on the SPECPower benchmark had an idle power of approximately 70% of its peak power.[24] This is of concern because, in this case, the power consumption is not proportional to workload; efficiency can be far below the peak efficiency of the server. Indeed, servers often spend much of the time at low utilization. "Energy proportional" scaling of energy use ensures that these servers will operate at high energy efficiency even at lower workload utilization.

Regulatory Environment

A significant outcome of the report to Congress was a focused effort by the Environmental Protection Agency's Energy Star program to create a standard for energy efficiency.[25] Since, at the time, the art of understanding and measuring server efficiency was nascent, initial efforts focused on measuring server idle power. As discussed earlier, idle power can be a good proxy for energy proportionality so long as server performance is also taken into account.

[24]See http://spec.org/power_ssj2008/results/res2007q4/power_ssj2008-20071129-00017.html.
[25]See www.energystar.gov/index.cfm?c=prod_development.server_efficiency_study.

www.dbooks.org

It's a common pitfall to equate energy efficiency uniquely with low power. Server idle power, while correlating in some cases to higher efficiency servers, cannot by itself be counted on as a reliable indicator of efficiency. The reason for this is that efficiency correlates to both server energy use and server performance. A computer with low performance will take relatively longer to complete a given amount of work, which can offset any benefits of reduced power.

The current Energy Star standard focuses broadly on energy efficiency, including efficient power supplies, capability to measure and monitor power usage, efficient components, and advanced power management features.[26]

In addition to the United States, several other countries have taken steps to encourage or even require certain levels of energy efficiency in servers. Among these are the European Union,[27] Australia,[28] and China. In some cases, energy efficiency restrictions are required due to a lack of necessary electrical grid capacity, whereas with other cases, the standards fit with a framework of reducing carbon footprint.[29]

A summary of international regulatory implications for server design is shown in Figure 1-4. Although server idle power is a common focus, approaches differ depending on location. This can be problematic since requirements for one (e.g., overall energy consumption) may not be consistent with another (e.g., computing energy efficiency). Server energy efficiency standards and regulations can focus on different aspects of energy efficiency. The Energy Star program focuses on idle power and component efficiency. It is planning to shift toward measures of energy efficiency.

	United States	Europe	China	Australia
Idle Power	☑	☑	☑	☑
Component Efficiency	☑			
Energy Use		☑	Under consideration	Under consideration
Computing Efficiency	In Development	Under consideration	Under consideration	

Figure 1-4. *Server energy efficiency standards and regulations*

[26]See www.energystar.gov/products/specs/enterprise_servers_specification_version_2_0_pd.

[27]See www.powerint.com/en/green-room/agencies/ec-eup-eco-directive.

[28]See www.energyrating.gov.au/wp-content/uploads/Energy_Rating_Documents/Product_Profiles/Other/Data_Centres/200905-data-centre-efficiency.pdf.

[29]See www.digitaleurope.org/DocumentDownload.aspx?Command=Core_Download&EntryId=109.

Efficient power supplies are important for overall server efficiency since any losses in the power supply are overhead for any energy uses ultimately for computation. In the 2006 timeframe, power supplies had efficiencies that were as low as 50%.[30] Low-efficiency power suppliers are cheap to produce, and since customers didn't demand higher efficiency, there was no incentive by the server manufacturer to improve efficiency. But the opportunity is enormous. With the adoption of 80 Plus power supply efficiency guidelines by the EPA for Energy Star in 2007, power supply efficiency rapidly improved. Current power supplies, to be Energy Star–compliant, are required to have efficiencies of 89% at 50% load and a power factor of 0.9. Comparing this to an efficiency of 50%, the power consumption of a server would be reduced 35% for a fixed load.

Measuring Energy Efficiency

It is a common pitfall to associate energy efficiency with low power. Efficiency generally associates a level of output for an amount of input. In the case of computing, the output associated with efficiency measurements is the number of computational cycles completed. Therefore, although low power can definitely contribute to energy efficiency, it is insufficient without adequate performance.

Several metrics are for measuring energy efficiency of servers, but two of the most common are SPECPower_ssj2008 and HPC Linpack. SPECPower was developed by the Standard Performance Evaluation Corporation (SPEC) in 2008 for the express purpose of measuring server energy efficiency. Linpack is a high-performance computing benchmark made up of a collection of Fortran subroutines.[31] It is used as a measure of energy efficiency on the Green500[32] listing of supercomputing energy efficiency.

SPECPower

SPECPower measures the efficiency of a single server using a graduated workload. The workload is graduated in increments of 10% of a measured maximum or 100% server workload performance. SPECPower is based on server-side Java, which has the advantage that measurements can be implemented with a single client set-up. Thus it is economical to operate.

An example output of published SPECPower measurement is shown in Figure 1-5.[33] Performance to power ratios are measured at an established set of points. The quantity

$$\sum ssj_ops / \sum power$$

is an accepted indicator of overall system energy efficiency. As of this writing (March 2015), measurement of over 480 systems have been published. The utility of published SPECPower data is very high since it separates the assessment of power and performance across what is all the "load line" from 0 to 100% of maximum workload.

[30]See http://en.wikipedia.org/wiki/80_Plus.
[31]See www.top500.org/project/linpack/.
[32]See www.green500.org/.
[33]See http://spec.org/power_ssj2008/results/res2013q4/power_ssj2008-20131001-00642.html.

www.dbooks.org

Benchmark Results Summary

Performance			Power	Performance to Power Ratio
Target Load	Actual Load	ssj_ops	Average Active Power (W)	
100%	99.7%	1,699,266	242	7,029
90%	90.1%	1,535,986	230	6,674
80%	80.2%	1,366,107	217	6,283
70%	70.0%	1,192,900	204	5,852
60%	60.0%	1,022,251	189	5,404
50%	50.2%	854,940	174	4,923
40%	40.0%	681,033	157	4,340
30%	30.0%	510,898	141	3,632
20%	20.0%	340,862	125	2,725
10%	10.0%	170,305	109	1,557
	Active Idle	0	79.6	0
			∑ssj_ops / ∑power =	5,019

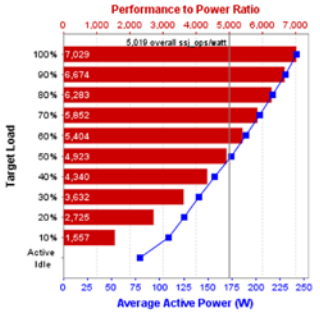


Figure 1-5. *A sample of a SPECPower published result. The table emphasizes both workload performance and energy efficiency*

The data published for SPECPower has shown a strong trend of improvement in the energy efficiency of servers. Although SPECPower is not measured for a large variety of servers, it is representative of the capability of servers whose power management is properly configured. Figure 1-6 shows a plot of the energy efficiency of all dual socket servers with Intel Xeon processors as a function of the "hardware available" data for the system. The data show that the energy efficiency of the servers are increasing exponentially (note the logarithmic scale), doubling approximately every 1.6 years. That means that in the 7 years since 2007 when the benchmark was published, energy efficiency has increase by about a factor of 20.

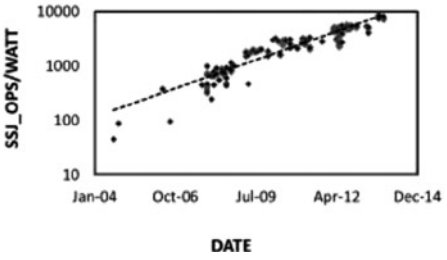


Figure 1-6. *Dual-socket server energy efficiency, as measured by SPECPower, Intel-Xeon based systems versus their "hardware available" date. Note the logarithmic scale, indicating an exponential trend*

What is less obvious is what the contributions are to the increase in energy efficiency. Since energy efficiency is a ratio of performance to power usage, the increase can be attributed to either a performance increase or a power decrease. It turns out both are responsible in the case of SPECPower.

To understand this, we can look at the details of the SPECPower data shown in Figure 1-7. The figure shows the trend of both the ratio of idle to maximum power and the performance for all published two-socket Intel Xeon–based servers at SPEC.org for the SPECPower_ssj2008 benchmark. Both trends emphasize the growing importance of energy-proportional behavior of servers in improving energy efficiency. The ratio of idle to max power is a metric for the proportionality of the server. SPECPower reports carry a wealth of information about the server, including CPU and memory configuration.

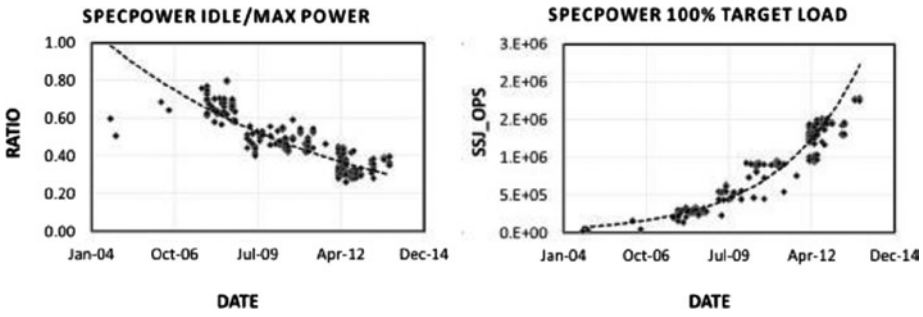


Figure 1-7. *Trend of both the ratio of idle to maximum power and the performance for all published two-socket Intel Xeon–based servers*

The historical trend of energy proportional efficiency can be visualized in another way—by examining the "load line" of respective generations of servers as measured by SPECPower. The load line is simply a graph of the server power versus the absolute workload. From the graph, the power, efficiency, and performance of the server can be deduced. Figure 1-8 shows the selected graphs from platforms built from specific generations of processor families. The horizontal axis measures computations work up to a measured system performance limit. The vertical axis measures system power. Over time, according to this specific benchmark, system performance has increased while system power has decreased.

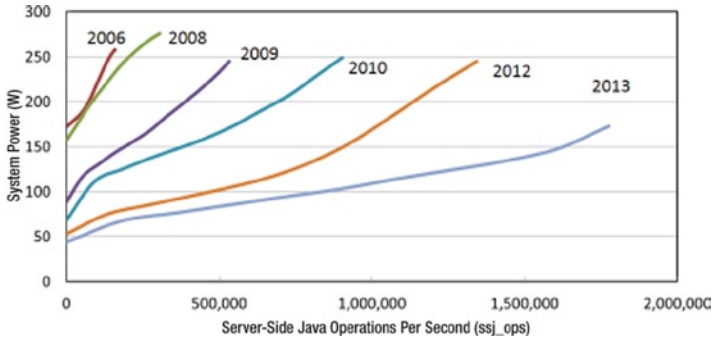


Figure 1-8. *The "load lines" of several generations of two socket servers as measured by SPECPower_ssj2008*

www.dbooks.org

How do you read the graph? System workload is plotted along the x-axis (from active idle to a load point of 100% system capacity) and system power is plotted along the y-axis. The curves for each server follow an intuitive progression; as system workload increases, power usage increases. The degree of that increase is related to the proportionality of the system. Note that higher performance is to the right, lower power is down, and therefore higher efficiency is to the lower right. Note also, work output capability is measured in *server-side Jave operations per second* or ssj_ops, which is a measure of system performance.

What's first evident from the graph is the higher peak performance in each successive generation. There is a gain in "peak" energy efficiency inherent with performance increases in the systems (more "work"). This is the progression known colloquially as Moore's Law. Note that the peak power of these systems is relatively constant at about 250 watts.

However, the graph reveals an additional progression toward lower power at low utilization, that is, toward delivering even higher gains in energy efficiency at actual data center workloads via "energy proportionality." Assuming each system is run at the mid-load point, the average power dropped from a little over 200 watts in 2006 to about 120 watts in 2012. That's a net power reduction of about 40% and, assuming $0.10/kWh energy costs and a PUE of 2.0, an operational cost saving of about $150/year. In addition, the work output capability (measured in ssj_ops) at that load point increases over a factor of 10.

The families of curves reveal several interesting trends. The first notable trend is the steady decrease in idle power of the systems. You'll notice the curves fall into sets of pairs. At a high level, this is because managing idle power of a server is primarily related to the microarchitecture. Indeed optimizing the features of the microarchitecture to achieve the right balance of power and performance capability is a main subject of this book.

You'll also note the steady increase in performance with each generation. These performance increases have two origins. In the years 2006, 2009, and 2012, new microarchitectures were introduced. In intervening years new process technologies were introduced (Intel's "tick-tock" model[34]) giving rise to lower power and also substantially increased performance. Table 1-1 lists the evolution of energy-efficient servers derived from both process technology and microarchitectural revolutions. Development of new architecture and new silicon process technologies represent huge investments in capital and engineering. The highlights emphasize the tick-tock development cycles of staggered process technology and architecture.

[34]See www.intel.com/content/www/us/en/silicon-innovations/intel-tick-tock-model-general.html.

Table 1-1. *The Evolution of Energy-Efficient Servers*

Year	Microarchitecture Family	Process Technology	Processor Family
2006	Core	45 nm	Xeon 5100
2008	Core	32 nm	Xeon 5400
2009	Nehalem	32 nm	Xeon 5500
2010	Nehalem	22 nm	Xeon 5600
2012	Sandy Bridge	22 nm	Xeon E5
2014	Haswell	14 nm	Xeon E5 v3

It is also instructive to look at the reduction in the energy per operation as deduced from the SPECPower data. The energy reduction is easily visualized in Figure 1-9 as the area of the rectangle defined by the average power and the time per ssj_op. Each data point is labeled for correspondence to Figure 1-8. The time per ssj_op is calculated as the reciprocal of measured ssj_ops at 10% utilization on the SPECPower trend curves in Figure 1-8.

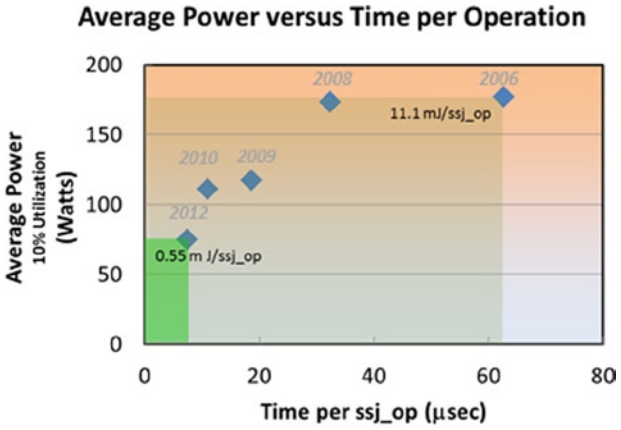


Figure 1-9. *A representation of the energy per ssj_op as measured by SPECPowerssj_2008 showing the role of both reducing the time and the power consumed while doing a computation. Both have been important in reducing overall energy consumption*

www.dbooks.org

What is interesting is the stair-step pattern shown in Figure 1-10—the trend of the energy per operation as a function of time shows a 41% per year reduction. From 2006 to 2008 we moved from 65 nm to 45 nm silicon technology, and from 2009 to 2010 from 45 nm to 32 nm silicon technology. In each case, the time to complete an operation decreased by about half. Complementing that, from 2008 to 2009, and from 2010 to 2012, were significant microarchitecture changes. These resulted in time reductions associated with performance gains, but also significant power reductions. Overall, both power and time reductions contributed to the gains in efficiency.

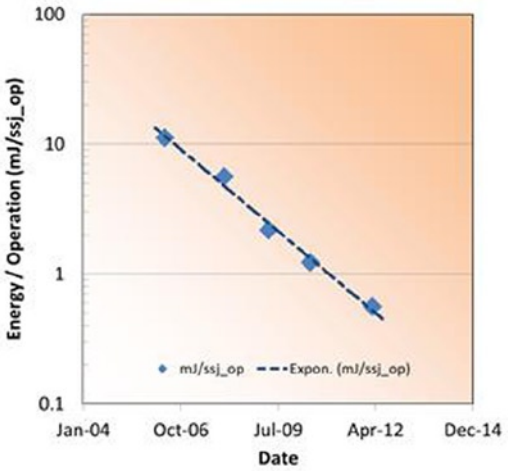


Figure 1-10. *The SPECPowerssj_2008 trend of the energy per operation as a function of time shows an exponential trend that is consistent with an efficiency-doubling time of 0.9 years. This is much faster than the 1.5 years reported by Koomey, owing to additional efficiency gains from energy proportionality*

Plotting the data as a time series versus the "system available" date from the SPECpower data shows the expected exponential trend. The fit parameters equate to a 41% per year reduction in the energy per operation and about a factor of 20 over the range shown. Putting the energy needed for computation into perspective, 0.5 milli-Joules is the energy needed to light a 100-watt bulb for about 5 microseconds.

The performance and efficiency gains from microarchitecture also play a strong role in other benchmarks, as the next discussion of high performance computing will show.

High Performance Computing Efficiency

High Performance Computing (HPC) is another area where a trend of computing efficiency has been established by well-accepted methods. The Green500 list has, since 2007, published a semi-annual list of the top energy-efficient super computers in the world.[35] The Green500 shares the same workload as the Top500 supercomputing performance list.[36] Both are based on HP Linpack, which derives from a collection of Fortran linear algebra routines written in Fortran in the 1970s. Excellent source material on the Linkpack routines can be found online.[37]

Alternative benchmarks have appeared, such as the Graph500,[38] which are more relevant to measuring performance of supercomputers running data-intensive applications. Arguably with the growth of "big data" applications to continue into the future, these kinds of benchmarks will be relevant to a broader range of supercomputing applications. However, at this writing, the alternatives are just getting going and have not yet gained the same recognition as have the Top500 and Green500 lists. As a result, this discussion will focus on the historical trends of the Green500 and Top500 lists.

At the scale of supercomputers today, performance leadership is practically inseparable from efficiency leadership due to the practical constraint of power. The power consumption of the largest supercomputers in the world is now between 10 and 20 megawatts. Although these limits are not written in stone, at an estimated infrastructure cost of about $10 per watt, the cost of expanding beyond those limits is prohibitive except for the largest governmental and private agencies. With the expanded role of supercomputing in everything from office scale DNA decoding to field-based geophysics, the need for higher performance in fixed-power environments is increasing.[39]

Since both performance and efficiency are important to supercomputing leadership, it is convenient to look at both the efficiency and performance of supercomputers simultaneously. The Exascalar method does exactly this, plotting the points from the Green500 list by their performance and efficiency.[40] Figure 1-11 shows the efficiency and performance of the computers in the Top500 supercomputer list since 2007. The historical trend line reveals that the performance gains of the top systems have been due to both efficiency gains and increases in power. Exascalar measures progress of supercomputing leadership toward a goal of 10^{18} flops (an *Exaflop*) in a power envelope of 20 megawatts. As is evident in Figure 1-11, the points fall roughly into a triangular shape with a taxonomy that reflects the state of the art in computing performance and efficiency and also cost.

[35]See www.green500.org/.
[36]See www.top500.org/.
[37]See www.top500.org/project/linpack/.
[38]See www.graph500.org/.
[39]See www.intel.com/content/www/us/en/research/tomorrow-project/intel-labs-dna-sequencing-and-bio-chem-sensing-video.html.
[40]See www.datacenterknowledge.com/archives/2012/07/10/june-2012-exascalar-efficiency-dominates-hpc/.

www.dbooks.org

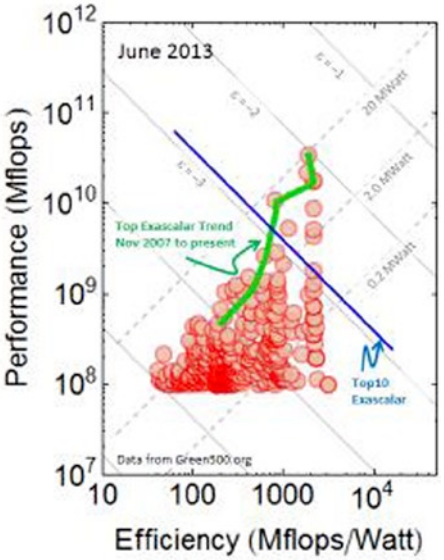


Figure 1-11. *The Exascalar plot of the June 2013 Green500 list*

The Exascalar values in this graph are computed from the formula where both efficiency and performance are normalized to the goal of one Exaflop in a 20 megawatt power envelope.

$$\varepsilon = \frac{1}{\sqrt{2}} Log\left(\frac{Efficiency}{10^{18}\ Flops/20MWatt} \frac{Performance}{10^{18}\ Flops} \right)$$

The factor of $\sqrt{2}$ ensures consistency with an earlier (but more complex and less generalizable) formulation of Exacalar.[41]

The earlier-mentioned triangular shape comes about because of the constraints of power in general application. Although the trend in increased power is evident from the trend line of the top Exascalar systems, that increase, about a factor to ten, also increases the installation costs by roughly a factor of ten and therefore represents a major barrier for a majority of adopters.[42] Another point to note in the graph is that systems in the lower left-hand corner consume almost 100 times the power of the systems in the lower right-hand corner of the triangle, but deliver the same performance. This represents a potentially very large difference in total cost of ownership (TCO).

[41]Balaji Subramaniam, Winston Saunders, Tom Scogland, Wu-chun Feng, "Trends in Energy-Efficient Computing: A Perspective from the Green500," Proceedings of the 4th International Green Computing Conference (Arlington, VA, June 2013).
[42]www.datacenterknowledge.com/archives/2013/01/28/the-taxonomy-of-exascalar/.

The trend of the Exascalar can also be plotted as a time series as shown in Figure 1-12. The top Exascalar system trend intersects the Exaflop equivalent of Exascalar ($\varepsilon = 0$) some time in the year 2019. The median Exascalar trend is increasing at a slower rate, which can be accounted for by the slower increase in power (but similar gains in efficiency) of the general population. The differential between the top and median Exascalar growth is accounted for by the increased power levels of the top systems.

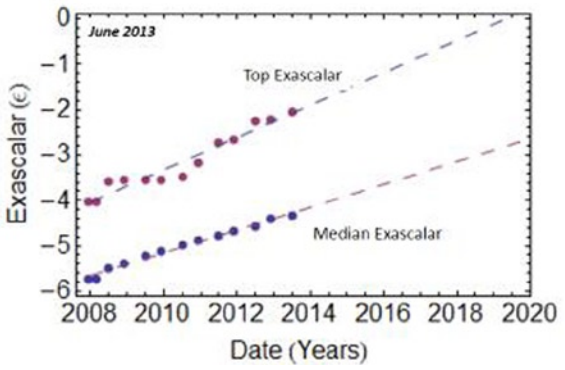


Figure 1-12. *The trend of the top and median Exascalar as a function of publication date*

Comparing theSPECPowerssj_2008 results with the Exascalar results shows the challenge of trending energy use and efficiency with benchmarks. In the case of SPECPowerssj_2008, the overall system power has decreased over time to benefit efficiency, while in the case of the HPC benchmarks, overall system power has increased over time to achieve higher performance.

Energy Efficiency and Cost

Energy efficiency is a highly desirable characteristic in data centers, but the overall goal of a data center is to meet the computational needs within both physical and financial constraints of the organization. These constraints are usually captured in a TCO model, which takes into account both capital and operational costs of the data center (see Table 1-2).

www.dbooks.org

TCO generally depends very strongly on the specific applications or intended use of the data center. This is a reflection of the wide range of applications for data centers. For instance, in some locations, the high costs of energy may favor the choice of a particular power envelope for the servers or, in some other cases, software licensing costs may strongly influence hardware choices.

However, outside these special cases, some general observations can be made about TCO.

Costs fall into two categories: capital costs and ongoing operational costs. The capital costs are associated with the the facility of the data center itself as well as the servers and other IT gear required to make the data center operate. Important operational costs include electricity, water, maintenance, and so on. Other factors, such as expected depreciation for both the facility and IT hardware, may also have a pronounced effect on the outcome of the model.

Many TCO models are available online. Some are made available for cost; some are available as a service.[43] These models have varying degrees of sophistication depending on the desired fidelity and tolerance for error.

Table 1-2 lists the ranges of parameters for a TCO model. The operational server energy cost includes overhead of PUE = 2.0. In both cases, the energy cost to run the servers in a data center is comparable to the facility cost itself.

Table 1-2. *Ranges of Parameters for a TCO Model*

	Low Cost Range (U.S.)	High Cost Range (U.S.)
Facility capital cost per watt	$8–$12	$20–$40
Facility capital depreciation	10 years	20 years
Facility capital cost/watt/year	$0.80–$1.20	$1.0–$2.0
Electricity cost per watt	$0.03/kWh	$0.15/kWh
PUE	1.2	2.0
Operational server energy cost/watt/year	$0.31	$2.62

Since the subject of this book is primarily server energy cost, a simplified model is shown in the table emphasizing the comparison of the facility cost with the energy needed to run the servers. The low cost range data center might correspond to an efficient cloud data center in a region selected for a mild climate and low-cost electricity. The high cost range might correspond to a highly secure and redundant data center near a major metropolitain area. In both cases, it is apparent that the energy costs of the data center are comparable to the facility capital cost.

[43] Vasileios Kontorinis, et al., "Managing Distributed UPS Energy for Effective Power Capping in Data Centers," *International Symposium on Computer Architecture, ISCA* (2012), http://cseweb.ucsd.edu/~tullsen/DCmodeling.html.

More sophisticated models take into account much more detailed analysis of individual data center costs, building upon and also substantiating the simpler analysis in Table 1-2.[44] In the model shown in Figure 1-13, power and cooling infrastructure costs are about equivalent to the utility energy costs. Although energy costs and facility capital costs represent about equal parts of the TCO, server depreciation is also an important contributor.

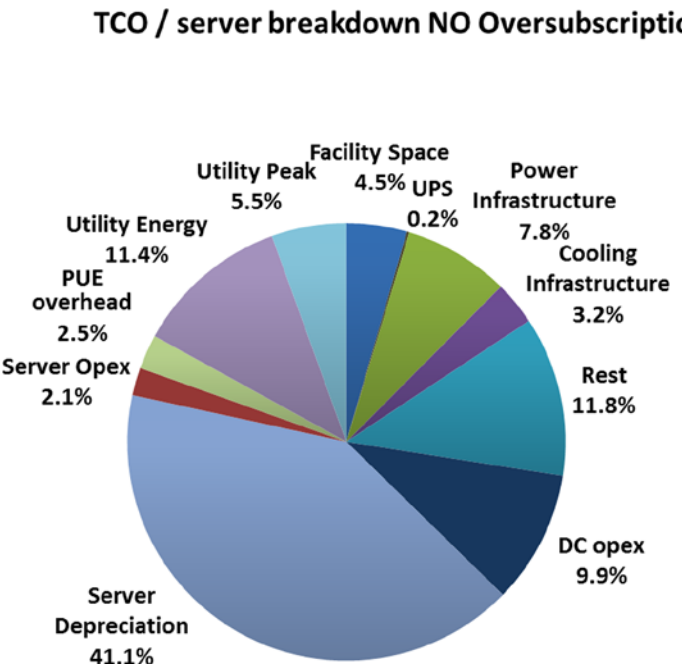


Figure 1-13. *An example of a breakdown of data center TCO*

However, traditional data center TCO models do not consider the cost of work output from the data center per se; they simply treat the servers as power-consuming units without regard for energy efficiency or performance of their computing capability. What is astonishing is that from a work output standpoint, the most wasteful energy consumers in data centers (even low PUE data centers) can be inefficient servers.

To illustrate this point, consider Figure 1-14, taken from an actual assessment of a Fortune100 company's data center. The analysis consisted of looking at the age distribution of the servers and then assessing, based on their configuration, energy consumption and finally their work output (or performance) capability. Although older servers were only 32% of the population, they consumed the majority of energy and only contributed a small fraction of the total computational output of the data center.

[44]Ibid.

www.dbooks.org

Since server efficiency doubles approximately every one to two years (depending on application and the specific metric used), older servers are far less efficient and constitute a larger fraction of energy use for a lower fraction of computing cycles.

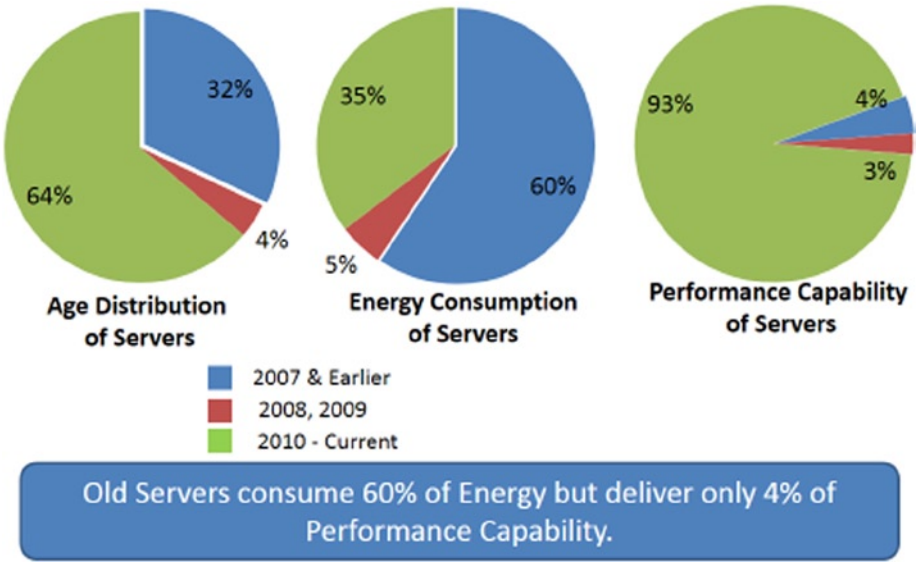


Figure 1-14. *Data from a walkthrough inventory of a Fortune 100 company showing the energy consumption and age distribution of servers*

In this particular data center, servers older than 2007 consume 60% of the energy but contribute only an estimated 4% of the compute capability. Although this may seem counterintuitive, consider the argument from the perspective of Moore's Law; if the performance doubles approximately every two years, servers from 2006 do approximately $1/8^{th}$ the computational work of servers dating from 2012, when the data was collected. Given the power consumption date presented earlier, it is also feasible that the energy consumption would decrease in newer servers, dependent on configuration.

Therefore, in data centers concerned not just about energy usage, but actual computational work, the energy efficiency and performance of the servers are important overall considerations. Detailed measurements on either actual or representative workloads are generally needed to achieve the highest levels of overall workload efficiency. The remainder of this book focuses specifically on the optimizations that can take place not only at the server level but also the data center level to optimize energy use and computational output of what may amount to a multi-million or even billion dollar investment.

Summary

In this chapter we have reviewed the performance and efficiency trends of data centers and have shown that the servers can contribute to the overall energy use in data centers, especially in cases where the efficiency of the infrastructure has been optimized.

We've compared the performance and efficiency trends of servers based on both the SPECPowerssj_2008 and the derived Exascalar benchmarks. In both cases, the efficiency of servers has improved exponentially over time, though with differing trends, depending on the specific workload.

In subsequent chapters, we will show how the efficiency of servers can be optimized for specific workloads, thus enabling users to tailor their server configurations for optimum performance and efficiency. In the final chapter of the book, we will tie these results back to TCO and show how performance, power, and cost tie together into an overall framework of datacenter TCO.

www.dbooks.org

CHAPTER 2

CPU Power Management

The CPUs and memory inside of a data center consume a fraction of the overall power, but their efficiency and built-in power management capabilities are one of the biggest influences on data center efficiency. Saving power inside of the CPU has multiplicative savings at larger scales. Saving 1 watt of power at the CPU can easily turn into 1.5 watts of savings due to power delivery efficiency losses inside the server, and up to 3 watts in the data center. Reducing CPU power reduces the cooling costs, since less heat must be removed from the overall system.

Before discussing how power is saved in the CPU, we will first review some basics of CPU architecture and how power is consumed inside of circuits. Then we will discuss the methods and algorithms for saving power inside of both memory and the CPU. Chapters 7 and 8 will investigate how to monitor and control these features.

Server CPU Architecture/Design

Over the years, server CPU core design has significantly evolved to provide high performance and energy-efficient execution of workloads. However, no core is complete without an effective support system to provide the core with the data it needs to execute. Caches, main memory, and hard drives provide a hierarchical mechanism for storing data with varied capacity, bandwidth, and latency tradeoffs. In more recent years, highly scalable interconnects have been developed inside CPUs in order to facilitate the scaling of the number of cores.

A less widely known goal of CPU design is optimization for total cost of ownership (TCO) amortization. Because the CPU plays a central role in information processing, matching the CPU with the right amount of performance/capabilities with the other data center infrastructure is critical to achieving the best TCO. Different workloads have

different sweet spots. For example, many high performance computing (HPC) workloads are very sensitive to scaling and cross-node communication. These communication networks can be very expensive and hence contribute significantly to data center TCO. In such systems, it is desirable to maximize per node performance in order to reduce the communication subsystem costs and dependency. On the other hand, a cold storage deployment[1]—where a large number of hard drives hold data that is very infrequently accessed over a connection with much lower bandwidth—may require much lower CPU performance in order to suit the needs of the end user.

CPU Architecture Building Blocks

Typical multi-core server CPUs follow a common high-level architecture in order to efficiently provide compute agents with the data that they require. The main components of a modern CPU are the cores that perform the computation, I/O for sending and receiving the data that is required for the computation, memory controllers, and support infrastructure allowing these other pieces to efficiently communicate with each other. Figure 2-1 shows an example of such a system. The boxes with a dashed outline are optionally included on the CPU Silicon die, whereas the others are now almost always integrated into the same die as the cores. Table 2-1 provides some high-level definitions for the primary CPU components.

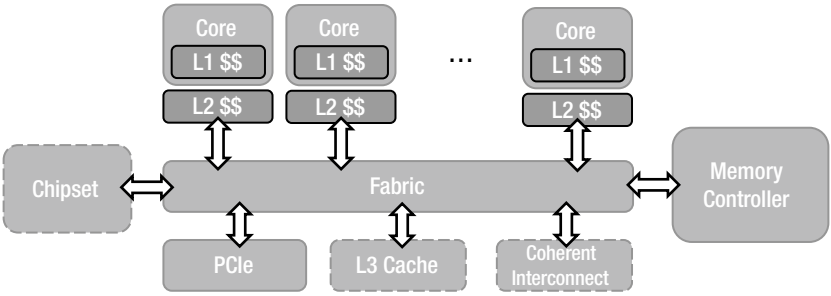


Figure 2-1. *A typical server CPU architecture block diagram*

[1]Cold storage is a usage model where a large amount of rarely used data is stored on a single system with a large number of connected hard drives to provide a massive level of storage.

www.dbooks.org

Table 2-1. *Primary CPU Components*

Component	Description
Core	Cores are the compute agents of a CPU. These can include general purpose cores as well as more targeted cores such as general-purpose computing on graphics processing units (GPGPUs). Cores take software programs and execute them through loads, stores, arithmetic, and control flow (branches).
Cache	Caches save frequently used data so that the cores do not need to go all the way to main memory to fetch the data that they need. A cache hierarchy provides multiple levels of caches, with lower levels being quick to access with smaller sizes, and higher levels being slower to access but providing much higher capacity. Caches are typically on the same die as the cores, but this is not strictly required (particularly with large caches).
On-die fabric	Interconnects exist on the CPU dies that are commonly called *on-die* or *on-chip* fabrics. These are not to be confused with fabrics that connect multiple CPU dies together at the data center level.
Memory controller	Memory controllers provide an interface to main memory (DDR in many recent processor generations).
PCIe	PCIe provides a mechanism to connect external devices such as network cards into the CPU.
Chipset	The chipset can be thought of as a support entity to the CPU. In addition to supporting the boot process, it can also provide additional capabilities such as PCIe, hard drive access, networking, and manageability. Chipset functionality is integrated into the same die or package as the cores in the microserver space.

Threads, Cores, and Modules

Traditional server CPUs, such as those found in Intel's Xeon E5 systems, are built using general purpose cores optimized to provide good performance across a wide range of workloads. However, achieving highest performance across a wide range of workloads has associated costs. As a result, more specialized cores are also possible. Some cores, for example, may sacrifice floating point performance in order to reduce area and cost. Others may add substantial vector throughput while sacrificing the ability to handle complex control flow.

Individual cores can support multiple *hardware threads* of execution. These are also known as *logical processors*. This technique has multiple names, including *simultaneous multithreading* (SMT) and *Hyper-Threading Technology* (HT). These technologies were introduced in Intel CPUs in 2002. SMT attempts to take advantage of the fact that a single thread of execution on a core does not, on many workloads, make use of all the resources

available in the core. This is particularly true when a thread is stalled for some reason (such as when it is waiting for a response from memory). Running multiple threads on a given core can reduce the per thread performance while increasing the overall throughput. SMT is typically a very power-efficient technique. The additional throughput and performance can increase the overall power draw, but the wall power increase is small compared to the potential performance upside.

■ **Note** There are two types of threads: hardware threads and software threads. Operating systems manage a large number of software threads and perform context switches to pick which software thread is active on a given hardware thread at a given point in time.

Intel Atom processors also have the concept of CPU modules. In these processors, two cores share a large L2 cache. The modules interface with the CPU fabric rather than the cores interfacing directly.

The terms *threads* and *processors* are commonly used to mean different things in hardware and software contexts. Different terms can be used to refer to the same things (see Table 2-2). This frequently leads to confusion.

Table 2-2. *Threads, Core, and Processor Terminology*

Term	Description
Hardware thread Logical processor Logical core	Hardware threads, logical processors, and logical cores are all the same. Each can execute a single software thread at a given point in time.
Hardware core Physical core	Hardware cores and physical cores represent a block of hardware that has the ability to execute applications. A single physical core can support multiple logical cores if it supports SMT. Logical cores that share a physical core share many of the hardware resources of that core (caches, arithmetic units, etc.).
Software thread	A software thread is a sequence of software instructions. Many software threads exist in a system at a given point in time. The operating system scheduler is responsible for selecting which software thread executes on a given logical processor at a certain point in time.

www.dbooks.org

Caches and the Cache Hierarchy

Server CPU cores typically consume a large percentage of the processor power and also make up a large percentage of the CPU area. These cores consume data as part of their execution. If starved for data, they can stall while waiting for data in order to execute an instruction, which is bad for both performance and power efficiency. Caches attempt to store frequently used data so that the core execution units can quickly access it to reduce these stalls.

Caches are typically built using SRAM cells. It is not uncommon for caches to consume as much area on the CPU as the cores. However, their contribution to power is much smaller since only a small percentage of the transistors toggle at any given time.

A range of cache hierarchies is possible. Figure 2-2 shows two examples of cache hierarchies. The figure on the left illustrates the cache hierarchy used on Xeon processors since the Nehalem[2] generation and the figure on the right illustrates the hierarchy used on the Avoton[3] generation. Different hierarchies have various performance tradeoffs and can also impact power management decisions. For example, the large L3 cache outside the cores in the design on the left may require the application of power management algorithms in order to achieve good power efficiency.

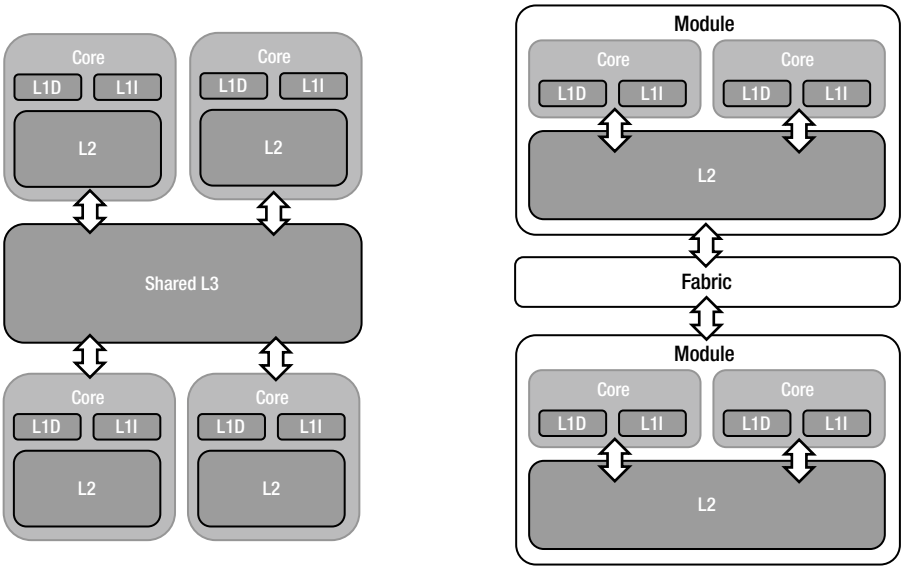


Figure 2-2. *Cache hierarchy examples*

[2]Nehalem is the code name for the Xeon server processor architecture released in 2008.

[3]Avoton is the code name for the Atom server processor architecture released in 2013.

Dies and Packages

CPUs are manufactured wafers of monocrystalline silicon. During manufacturing, each wafer is printed with a large number of rectangular CPU dies that are subsequently cut from the wafer once the manufacturing is complete. A moderately large server die is on the order of ~20 mm on a side (~400 mm^2). Figure 2-3 shows two magnified dies, one from the 8c Avoton SoC (system on a chip) and another from the Ivy Bridge 10c. The Avoton die is actually much smaller in size than the Xeon.

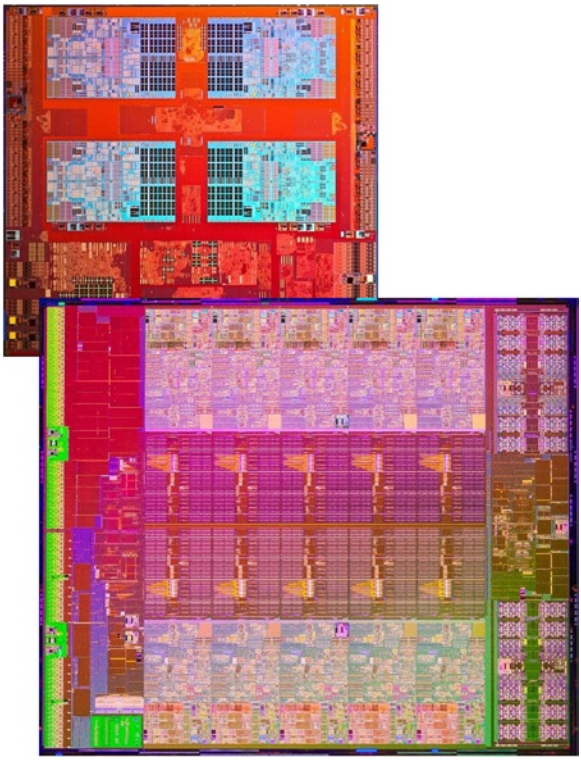


Figure 2-3. *Die photos of the 8c Atom Avoton (top) and 10c Xeon Ivy Bridge EP (bottom) (not to scale)*

Dies are then placed into a *package* as part of the manufacturing process. The package provides the interface between the die and the motherboard. Some packages (particularly lower power and lower cost offerings) are soldered directly to the motherboard. Others are said to be *socketed*, which means that they can be installed, removed, and replaced for the motherboard. The package connects to the motherboard through metal *pins*, which provide both power to the CPU and communication channels (such as the connection to DDR memory). Power flows into a CPU through many pins, and higher power CPUs require more pins in order to supply the required power. Additional connectivity (such as more DDR channels or support for more PCIe devices) also increases pin count.

www.dbooks.org

Packages can also include an *integrated heat spreader* (IHS), which is conceptually an integrated heat sink. Removing heat generated by the consumption of power within a CPU is critical to achieving high performance systems. IHSs help to spread the heat from the cores (and other areas with high power/heat density) out to the rest of the die to avoid hot spots that can lead to early throttling and lower performance. Figure 2-4 shows two CPU packages—one from an Avoton SoC and one from a Sandy Bridge. The Sandy Bridge package is much wider and deeper to accommodate the larger die and additional pins, but is also much taller. Part of this additional height is due to the IHS.

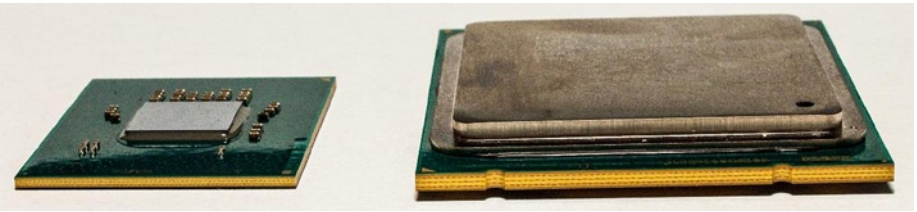


Figure 2-4. *Package photos of an 8c Xeon Sandy Bridge EP (right) and 8c Atom Avoton (left)*

Multiple dies can be included in a single package. This is called a *multi-chip package* (MCP). MCPs can provide a cost-effective way for increasing the capabilities of a product. One can connect two identical dies (commonly used to increase core count), or different dies (such as a chipset and a CPU). Connecting two devices inside of a package is denser, lower power, and lower latency than connecting two separate packages. It is also possible to connect dies from different process technologies or optimization points. MCPs have been effectively used in the past to provide high core count processors for high-end servers without the need for huge dies that can be cost prohibitive to manufacture. Dies within an MCP share power delivery and thermal constraints with each other, and therefore there are limits. For example, it can be very challenging (and expensive) to cool two 130 W CPUs stuck together into a single 260 W package. Bandwidth and latency between two dies in an MCP are also constrained compared to what is possible in a single die.

On-die Fabrics and the Uncore

Historically, Intel has referred to all of the on-die logic outside of the cores as the *uncore*. In the Nehalem generation, this included the L3 cache, integrated memory controller, QuickPath Interconnect (QPI; for multi-socket communication), and an interconnect that tied it all together. In the Sandy Bridge generation, PCIe was integrated into the CPU uncore. The uncore continues to incorporate more and more capabilities and functionality, as additional components continue to be integrated into the CPU dies. As a result, the CPU is now being replaced with the concept of system on a chip (SoC). This is most common in user devices such as cell phones, where a large number of special-function hardware components provide various capabilities (modems, sensor hubs, general purpose cores, graphics cores, etc.). It is also spreading into the server space with products like Avoton that incorporate cores, SATA, Ethernet, PCIe, USB, and the chipset into a single CPU package. Increased integration can reduce TCO because fewer discrete

devices must be purchased. It can also result in denser designs for the same reason. It can also be more power efficient to incorporate more functionality into a single die or package as higher performance connections consume lower power when integrated.

In these SoCs, the interconnect that provides the communication between the various IPs has been termed an on-die *fabric* in recent years. Off-chip fabrics that connect multiple CPUs together into large, non-coherent[4] groups of CPUs also exist. Modern on-die fabrics are the evolution of the uncore interconnect from earlier generation CPUs.

On servers, when the cores are active and executing workloads, the power contribution from the uncore tends to be much smaller than the cores. However, when the cores are all idle and in a deep sleep state, the uncore tends to be the dominant consumer of power on the CPU as it is more challenging to efficiently perform power management without impacting the performance of server workloads. The exact breakdown of power between the cores and uncore can vary widely based on the workload, product, or power envelope.

Power Control Unit

As power management has become more and more complex, CPUs have added internal microcontrollers that have special firmware for managing the CPU power management flows. At Intel, these microcontrollers are called both the *PCU* (power control unit) and the *P-Unit*, and the code that they execute is called *pcode*. The PCU is integrated into the CPU with the cores. These microcontrollers are generally proprietary, and the firmware that runs on them is kept secret. It is not possible for OEMs or end users to write their own firmware or change the existing firmware in these PCUs. However, various configuration options are available to the OEM and end user. These can be controlled through either the OS or BIOS. Tuning and configuring these options is discussed in Chapter 8.

The PCU is responsible for the bulk of the power and thermal management capabilities that will be discussed through the rest of this chapter. The firmware running on the microcontroller implements various control algorithms for managing the power and performance of the CPU. Table 2-3 provides a high-level snapshot of some of the roles and capabilities of the PCU. The PCU is connected to almost every major block of logic on the CPU die and is continuously monitoring and controlling their activity.

Table 2-3. *Common PCU Roles*

Role	Description
Power management	Central control center for managing voltage, frequency, and other power saving states
Thermal management	Implements algorithms to prevent the CPU from overheating
Reset controller	Facilitates powering up the CPU

[4]Coherent fabrics are also possible and are traditionally used in supercomputer designs.

www.dbooks.org

Firmware can be patched in the field, either through BIOS or even directly from a running system in the OS. However, patch deployment after devices enter production is not frequent.

Server vendors do use their own proprietary firmware that runs off-chip on a *baseboard management controller* (BMC; a small microcontroller). This firmware frequently interacts with the PCU for performing both power and thermal management through the *Platform Environment Control Interface* (PECI). These topics will be discussed in more detail in Chapters 4 and 5.

External Communication

Although performing calculations is important on CPUs, getting data in and out of the CPU is a key part of many server workloads. Table 2-4 provides an overview of a selection of the key interfaces.

Table 2-4. *External Communication*

Interface	Details
Memory (DDR)	Memory provides storage for application code and data. It can also provide caching for frequently accessed data from drives. It is not uncommon for server CPUs to have hundreds of GBs of memory capacity, and even TBs are possible.
Drive storage	Drive storage is also common on servers. Some end users are moving away from having any local drive storage on compute nodes, electing instead to store all persistent data on separate storage nodes that are accessed over a high bandwidth network. The boot process can even be performed completely over the network. This can save significant procurement cost. Other customers still find a need for local storage on individual nodes.
Networking	Ethernet or InfiniBand are staples of most server nodes for moving data in and out of a given CPU for processing, or between nodes for tasks that utilize multiple CPUs for a single task.
Video ports	Video ports are rare and generally are not included on the platform. It is common for users to connect discrete graphics cards in the rare occasion where video is required.
USB ports	USB ports are also common and are primarily used for special tasks like firmware updates or debugging (not during normal execution).
Manageability	Servers commonly include an interface like PECI for external controllers to manage the server. These interfaces provide a mechanism for tasks like monitoring temperature or controlling power without interfering with the software running on the CPU cores.

(*continued*)

Table 2-4. (*continued*)

Interface	Details
Coherent interconnects	In deployments that have multiple CPUs per node, a coherent interconnect is used to connect the multiple sockets (e.g., Intel QPI). This allows multiple CPUs to be connected to each other and share a single operating system.
Non-coherent bridging	Some CPUs also support technologies to create non-coherent interconnects between nodes using PCIe (e.g., Intel NTB [Non-Transparent Bridge]). These technologies create non-coherent "windows" into the physical memory space across two machines where each machine appears as a PCIe device to the other machine (with a memory-mapped I/O [MMIO] range assigned to it). Today it is primarily used in storage usage models for redundancy across servers.

Thermal Design

CPUs consume power in order to execute; that power must be dissipated in order to keep temperatures under control. On modern CPUs, thermal sensors exist to monitor the temperature and help guarantee that the CPU will not get to a dangerous temperature where reduced reliability or damage could occur. CPUs may throttle themselves to stay under a target temperature or even initiate an immediate shutdown if temperature exceeds certain thresholds.

Most server CPUs are sold with *thermal design point* (TDP) power. The TDP specifies the amount of power that the CPU can consume, running a commercially available worst-case SSE application over a significant period of time and therefore the amount of heat that the platform designer must be able to remove in order to avoid thermal throttling conditions. The TDP power is generally paired with a *base frequency* (sometimes called the *P1 frequency*). A defined TDP condition is used to characterize this (power, frequency) pair. The goal of the TDP condition on servers has been to identify the worst-case real workload[5] that a customer may run. Different vendors (or even different products from the same vendor) can use varied TDP definitions, making it difficult to use this number for meaningful comparisons across these boundaries. Sequences of code that will consume more power at the TDP frequency than the TDP power do exist, and these workloads will be throttled in order to stay within the design constraints of the system and to prevent damage to the CPU.

■ **Note** Different workloads can consume a wide range of power at the same frequency. Many workloads consume significantly lower power than the TDP workload at the TDP frequency. *Turbo* is a feature that allows those workloads to run at higher frequencies while staying within the thermal and electrical specifications of the processor.

[5]AVX applications are not included in the base frequency on current server processors. Starting with HSW E5, a secondary "AVX P1" frequency was provided with each SKU to provide guidance for high-power AVX workloads.

www.dbooks.org

Many traditional server processors have had TDP power in the range of ~30 W to ~150 W. Microservers push TDPs much lower—down to ~5 W. Although it is possible to build processors with larger TDPs, these tend to be more challenging to work with. Larger heat densities can be difficult to cool efficiently and make cost effective. It is also possible to have larger processor dies that have less heat density, but these dies can also be challenging to manufacture efficiently.

Some client processors have adopted a concept called *Scenario Design Power* (SDP). This concept suggests that designing for the TDP may result in over-design in certain usage models. SDP attempts to provide OEMs with guidance about the thermal needs of certain constrained usage models. SDP has not been adopted for any server products at this time. Servers tend to rely on Turbo to reduce exposure to any platform over-design caused by designing to TDP.

CPU Design Building Blocks

The CPU architecture is constructed with a mix of analog and digital components. Analog design is typically used for designing the off-chip communication (such as the circuits that implement PCIe and DDR I/O), whereas the bulk of the remaining system is built out of digital logic.

Digital Synchronous Logic and Clocks

The bulk of the computation performed by CPUs is done by *digital synchronous logic*. Synchronous designs can be thought of as large pipelines. Tasks are broken up into subsets of work (see Figure 2-5). Groups of logic gates (implemented with transistors) take input data (1s and 0s) and calculate a set of output data. It takes time for the transistors to compute the answer from an input set, and during that time, it is desirable for that input data to be stable. *Flops* store state for logic while it computes and store the output data when it is ready for the next set of logic. Clocks, distributed throughout the CPU, tell these flops when they should *latch* the data coming into them.

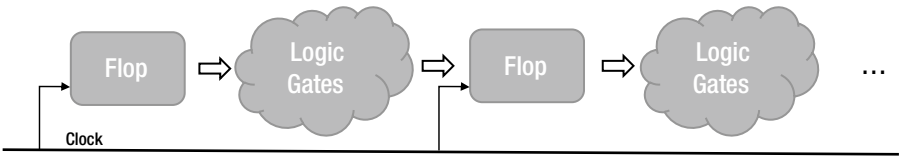


Figure 2-5. *Digital synchronous logic*

When people think of CPUs, they generally think about all the logic inside the CPUs that conceptually does all the work. However, clocks are necessary to all these digital circuits and are spread throughout the CPU. Clocks are typically driven by phased-locked loops (PLLs), although it is also possible to use other simpler circuits, such as a ring oscillators. Many modern PLL designs provide configurability that allows them to be locked at different frequencies. It takes time (generally measured in microseconds) to lock a PLL at a target frequency.

SRAM and eDRAM

Static random-access memory (SRAM or static RAM) is a block of logic that is used to store data. Most caches are built based on SRAM designs, and therefore SRAM commonly makes up a large percentage of the CPU die. Dynamic RAM (DRAM) is another type of logic that can be used to store data, and it is used for DDR devices.

SRAM is much larger in size than DRAM and consumes more power per byte of data, but it is also much faster to access and easier to design with. Unlike DRAM, it is built with similar manufacturing techniques to normal CPU logic, making it more amenable to integration into a single CPU die with other logic. It is possible to build large caches using embedded DRAM (eDRAM). eDRAM is used in Haswell E3 servers.

I/O

I/O circuits provide the capabilities for communication on and off a die. For example, these circuits are used for DDR, PCIe, and coherent interconnects like QPI. Most interconnects are parallel—transmitting multiple bits of data simultaneously. However, some serial interconnects still exist for low-bandwidth communication with various platform agents like voltage regulators.

There are two main types of I/O that can be used: *differential signaling* and *single-ended signaling*. Single-ended signaling is the simplest method for communicating with I/O. Conceptually, to transmit N bits of parallel data, $N + 1$ wires are required. One wire holds a reference voltage (commonly 0 V ground) whereas the others transmit binary data with a predefined higher voltage representing a 1. Differential signaling is more complicated, using a pair of wires (called a differential pair) to transmit a single bit of data. Differential signaling is less exposed to noise and other transmission issues, and therefore it provides a mechanism to reach higher frequencies and transmission rates. However, differential signaling requires roughly twice the platform routing compared to single-ended signaling and also tend to consume more power—even when they are not actively transmitting useful data.

Intel Server Processors

Throughout this chapter, various recent Intel server processors will be referred to by their codenames in an attempt to illustrate the progression of the technologies. Figure 2-6 illustrates the progression of the Intel server processors. Each major server processor generation is shown in a box with its major characteristics (number of supported sockets, number of cores, and process technology). Groups of processors with similar architectures have been grouped together with different shades of gray boxes. As an

www.dbooks.org

example, the Sandy Bridge-E5, Ivy Bridge-E5, and Ivy Bridge-E7 processors are all based on a similar architecture, which is separate from the single-socket Sandy Bridge-E3 and Ivy Bridge-E3 processors.

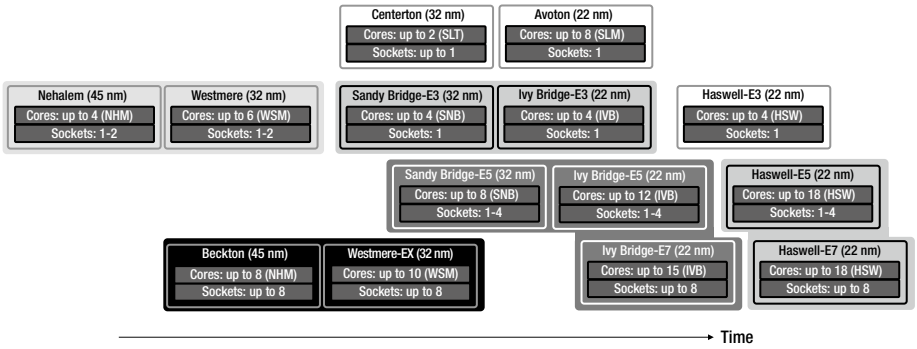


Figure 2-6. *Intel server processor progression*

It is important to note that the E3 products are based on desktop processor architecture and are therefore limited to a single socket and lower core counts. At the same time, they have much earlier time to market than the E5 and E7 processors. So, although a Haswell-E3 and Haswell-E5 share the same core design, the uncore design is different.

Introduction to Power

One of the first topics taught in electrical engineering is Power = Current * Voltage (P = I * V). You can think of power as a pipe with water flowing through it. Current is effectively how fast the water is flowing, whereas the voltage is the size of the pipe. If you have a small pipe (low voltage), it is difficult to move a lot of water (electricity). Similarly, if you can slow down how fast the water flows, you can reduce your water usage. Power management in a CPU is all about efficiently (and dynamically) controlling both current and voltage in order to minimize power while providing the performance that is desired by the end user.

Figure 2-7 illustrates a conceptual hierarchy of where power goes from the wall down to the circuits inside the CPU. This section will primarily explore the CPU and memory power components.

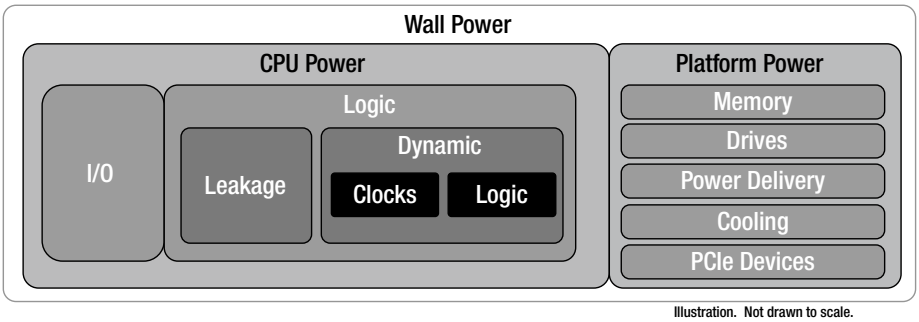


Figure 2-7. *Wall power breakdown illustration*

Table 2-5 provides a summary of some common power terms.

Table 2-5. *Common Power Terms*

Term	Abbreviation	Description
Voltage	V	Voltage is the electrical potential difference between two points.
Current (amps)	A	Current is the rate at which the energy flows.
Capacitance	C	Capacitance is the ability of a system to store an electrical charge. Batteries can be thought of as large capacitors that store charge.
Frequency	f	Frequency refers to number of transitions in a unit of time. In processors, this generally refers to the rate at which the clock is toggling.
Energy (joules)	J	Joules are a unit of energy or work. It does not matter how fast or slow the work is done—just how much work it takes.
Power (watts)	W	Power is a measurement of energy over time. Doing the same amount of work in half the time requires twice the power.

CPU Power Breakdown

The CPU power can conceptually be broken into

- The logic power (executing the instructions)
- The I/O power (connecting the CPU to the outside world)

These can be broken down further as described in the following sections.

www.dbooks.org

Logic Power

When the logic in the CPU transitions between 0 and 1, power is consumed. The transistors are effectively each little tiny capacitors that are charging and discharging (and expending power in the process). This is referred to as the *active power* of the CPU.

There are two components to active power:

- Power consumed by the clocks that run throughout the CPU.
- Power consumed by the actual logic that is performing computation.

Only a subset of the bits in the CPU transition between 0 and 1 in a given cycle. Different workloads exhibit different switching rates. This leads to the application ratio (AR) value in the equation, which modulates the active power. For example, it is common for certain types of workloads to not perform floating point math. In these workloads, the floating point logic is unused and will not transition and consume active power.

Leakage power can be thought of as the charge that is lost inside of the CPU to keep the transistors powered on. The equations for leakage are more complicated than for active power, but conceptually it is simple: leakage power increases exponentially with both voltage and temperature.

The breakdown between leakage and dynamic power is very sensitive to the workload, processor, process generation, and operating conditions. Dynamic power typically contributes a larger percentage of the CPU power, particularly when the processor is running at a high utilization.

Table 2-6 summarizes the CPU logic power breakdown.

Table 2-6. *CPU Logic Power Breakdown*

Component	Conceptual Equations	Description
Active power	$I \sim C * V * f * AR$ $P \sim C * V^2 * f * AR$	Active power can be thought of as the power consumed to toggle transistors between 1s and 0s.
Leakage power	$I \sim e^V * e^t$ $P \sim V * (e^V * e^t)$	Leakage power can be thought of as the charge that is lost inside of the CPU to keep the transistors powered on.

I/O Power

Running high bandwidth interconnects that are common in modern CPU designs can contribute a large percentage of the CPU power. This is particularly true in the emerging low-power microserver space. In some of these products, the percentage of power consumed on I/O devices tends to be a larger percentage of the overall SoC power.

There are conceptually two types of I/O devices: those that consume power in a manner that is proportional to the amount of bandwidth that they are transmitting (DDR), and those that consume an (almost) constant power when awake (PCIe/QPI).

I/O interfaces also have active and leakage power, but it is useful to separate them out for power management discussions. The switching rate in traditional I/O interfaces is directly proportional to the bandwidth of data flowing through that interconnect.

In order to transmit data at very high frequencies, many modern I/O devices have moved to *differential signaling*. A pair of physical wires is used to communicate a single piece of information. In addition to using multiple wires to transmit a single bit of data, typically the protocols for these lanes are designed to toggle frequently and continuously in order to improve signal integrity. As a result, even at low utilizations, the bits continue to toggle, making the power largely insensitive to bandwidth.

Table 2-7 summarizes the types of I/O power.

Table 2-7. *Types of I/O Power*

Component	Conceptual Equations	Description
Traditional I/O power	$I \sim BW * V * f$	Traditional I/O components typically exhibit power utilization that is a function of their bandwidth (utilization) along with voltage and frequency. Example: DDR3/4 data and command busses
Differential signaling I/O power	$I \sim V * f$	Differential signaling I/O power is a function of voltage and frequency but is generally not sensitive to bandwidth (utilization). Examples: PCIe, Ethernet, and Intel QPI all use differential signaling to transmit data.

Frequency, Voltage, and Temperature Interactions

Although power can easily be thought of as a function of voltage, frequency, and temperature, each of these components has an impact on the way that the others behave. Thus, their interaction with each other is also of relevance to energy efficiency.

In order to increase the frequency of a system, you must also increase the voltage. The voltage required to run a circuit tends to increase with the square of the frequency (see Figure 2-8). This relationship is critical to power efficiency and understanding power management. At some low frequencies, it is possible to change the frequency with only a small (if any) impact to voltage and relatively small increases in overall power. At higher frequencies, a large increase in voltage is required to get just a small increase in frequency. The exact relationship between these two components is based on the transistor design. There are varied manufacturing and design techniques that are used to select the operating voltage at different frequency points. So, although conceptually voltage scales with the square of the frequency, this is not always how real systems operate in production.

www.dbooks.org

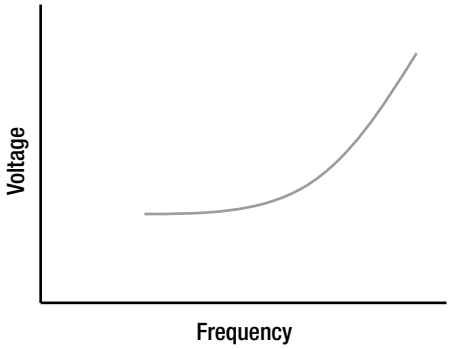


Figure 2-8. *Voltage/frequency relationships*

Different transistor designs and process technologies have different characteristics. Transistors that can achieve higher frequencies must trade off low-power characteristics. These are commonly used in high-power server CPU designs. On the other hand, transistors can be optimized for low leakage and low-power operation, trading off high frequency operation. This type of transistor is used in phone, tablet, and laptop devices. They can also be used in microservers and other low-power servers. Both types of transistors can be used to build power efficient CPUs and data centers.

■ **Note** Executing at a lower voltage and frequency (and power) does not necessarily make a system more power efficient. Rather, the most efficient operating point tends to exist around the "knee" of the exponential curve (or slightly to the right of the knee). A common misconception is that the lower the frequency and the lower the power, the more efficient the operation. This is commonly incorrect, particularly when power is measured at the wall. It is also possible to build very power efficient data centers using both low-power CPUs leveraging power-optimized transistors and higher power CPUs based on frequency optimized transistors.

Leakage current is exponentially sensitive to temperature. Traditionally, increases in temperature have resulted in higher power as a result of increases in leakage current. However, leakage power has trended down in recent process generations. The result is that there is less sensitivity to temperature.

There is another phenomenon called *inverse temperature dependence* (ITD). As temperature goes down, the voltage required to operate a transistor at a given frequency can increase. This behavior is most pronounced at lower voltages. In high-power server CPUs, this phenomenon typically does not impact peak performance or power, since voltage and temperature in these situations are high enough that there is minimal if any ITD compensation required. However, ITD can become more significant

in low-power CPUs that operate at lower voltages, frequencies, and temperatures. The ITD phenomenon has been known for many years but may become more notable as leakage power is driven down. Historically, as temperatures decreased, leakage power dropped more than the increase in power from ITD. On products with very low levels of leakage power, ITD effects could result in increased net power at low temperatures.

Power-Saving Techniques

Now that we have looked at the basics of where power goes in the data center, we will investigate some of the high-level techniques for achieving power efficiency. There are two conceptual ways to save power:

- Turn it off.
- Turn it down.

Different components in the data center and CPU have different techniques for performing each of these two operations. The rest of this chapter will go into some of the details of those techniques.

Turn It Off

Turning off the lights in your house is a very effective way to save power. When CFL light bulbs first were introduced to the market, many were unhappy with the long time it took for them to provide the desired amount of light quickly. In a CPU, similar issues arise. There are different levels of "off," and the tradeoffs are made between saving power and how quickly different subcomponents are available when desired (see Table 2-8).

Table 2-8. *Turning Logic Power Off*

Component	Wake Latency	Description
Clock gating	~10 ns to ~1 µs	Stop the clocks, saving active power
Power gating	~1 to 10 µs	Removes all power, saving both leakage and active power

Synchronous design used in modern CPUs depends on clocks to be routed throughout the logic. If a given block of logic is not in use, the clocks going to that logic do not need to be driven. *Clock gating* is the act of stopping the clocks to a given block of logic to save power. By gating the clocks, both the power of the clocks themselves can be saved, as well as any other dynamic power in the logic (since it cannot transition without clocks).

Clock gating can be performed at a wide range of granularities. For example, a single adder could be clock gated if not in use, or an entire core could be clock gated. Clock gating can be performed autonomously by the hardware when it detects logic is not in use, or it can be performed with software intervention. When clocks to a block of logic are gated, the dynamic power of that block is driven down close to zero, whereas the leakage power is not impacted. State (information) in the circuit is maintained.

www.dbooks.org

Power gating is a technique that allows both leakage and active power to be saved. However, it takes much longer to wake the circuits back up compared to clock gating. In addition to preventing transistor state transitions, power gating removes all power from a circuit so that leakage power is also driven to zero. State is lost with power gating, so special actions (like save/restore or retention flops) must be used in conjunction with power gating.

Turn It Down

Voltage has a significant impact on both the dynamic and leakage power of a circuit. By reducing the voltage when performance is not required, power can be saved. Table 2-9 provides a summary of two common mechanisms for reducing voltage.

Table 2-9. *Turning Logic Power Down by Reducing Voltage*

Component	Description
Voltage/frequency scaling	If high frequency is not required, it can be dynamically reduced in order to achieve a lower power level. When frequency is reduced, it may also be possible to reduce the voltage.
Retention voltage (Vret)	The voltage required to maintain state in a circuit can be lower than the voltage required to operate that circuit. For example, maintaining data in a cache can be done with much lower voltage than is required to read/write that data. Decreasing the voltage to Vret is frequently paired with clock gating in order to achieve a "middle ground" between basic clock gating and power gating. Compared to power gating, some leakage power continues to be consumed, but state is maintained allowing for simpler designs and faster wake latencies.

■ **Note** Voltage reduction is a critical piece to power savings. Leakage power scales exponentially with voltage, and dynamic power scales about with the square of the voltage.

Power-Saving Strategies

One major challenge with power management algorithms is understanding how multiple algorithms will impact one another. Saving power comes with some cost. For example, if you put memory into a low-power state, it takes time to wake it back up in order to service a memory request. While that request is waiting for memory to wake back up, something else in the system is generally awake and waiting while consuming energy. Aggressively saving power in one part of the system can actually result in a net power increase in the

overall system if not done carefully. Features can be enabled that save power for their subsystem at some overall performance cost and minimal to no overall power savings. A good system design will hide these challenges from the end users and enable them to get the most out of their system.

The platform characteristics can play a large role in determining "what's best." As an example, in a system with 1 TB of memory connected across two sockets, a large percentage of the platform power is spent in the memory. Aggressively using memory power management here is generally a great idea. On the other hand, if a system only has 8 GB of memory and a single DIMM of memory, using memory power management can only save a small amount of overall memory power and may increase platform power in certain conditions because of increase active time in the IA cores. Chapter 8 will discuss some of these tuning options and tradeoffs.

Race to Idle vs. Slow Down

When going on a road trip, cars are traditionally most efficient when running at about 60 mph. If you drive faster than that, the car will be active for a shorter amount of time, its efficiency while active will be less, and it will consume more gas. If you drive slower, gas may be consumed at a slower rate (in time), but the overall gas spent will be larger because the car is active longer. At speeds higher than 60 mph, there is higher wind resistance and drag on the car, and engines are typically not optimized to run as efficiently. At lower speeds, the drag may be lower, but the engine is running below its capabilities, making it less efficient.

Similar behavior can exist inside of a CPU. The speed of the car is similar to the voltage/frequency of the CPU. Theoretically, you can achieve the best power efficiency by cycling between the most efficient operating point and turning it off in order to supply the desired level of performance. This strategy has traditionally been referred to as *Race to Idle* or *Race to Halt* (`HALT` is a CPU instruction instructing a core to stop executing and go into a power saving state).

The Race to Idle strategy has generally been shown to be inefficient in many server usage models because the idle state consumes too much power due to its constraints. Imagine that it would take one hour to start your car whenever you wanted to use it. If you were using your car frequently throughout the day, you would just never turn the car off. At night, it might be a great idea, but on a weekend filled with chores, you would be unwilling to wait for your car to warm up. Similarly, a commuter with a fixed schedule might be able to tolerate taking one hour to turn on their car in the morning (they could turn it on before getting ready for work). This is because they know when they are going to need it. A doctor who is "on call," on the other hand, would not be able to tolerate this because they may need to go into work at any time and would have zero tolerance for a delay. So, even if they are able to rush through their tasks, they would have to leave the car running when they were done with it.

Servers tend to be more like on-call doctors. They never know exactly when they are going to be needed, and they need to be available quickly when they are needed. Problems like network packet drops can occur if deep idle states are employed that require long exit latencies. At times, servers know that they will not be needed (i.e., the doctor goes on vacation). However, this is generally the exception rather than the common case.

www.dbooks.org

Table 2-10 summarizes some of the different techniques that can be used to save power in a server CPU.

Table 2-10. *Power-Saving Strategy*

Strategy	Driving Example	Server Application
Race to Idle	Drive 100 mph taking rest stops	The server runs at peak power and performance in an attempt to get into a deep power saving state. This typically is not effective or employed in server usage models. Too much power is consumed in idle states to make this effective, because very deep idle states take too long to wake up. It is difficult to predict when to wake up accurately.
Jog to Idle	Drive 60 mph taking rest stops	The server runs at an efficient operating point that is still slightly faster than required at a given point in time and then attempts to get into an idle state. This technique theoretically sounds good, but actually achieving periods of idleness is challenging.
Slow and Steady	Drive 45 mph continuously	The server runs at the utilization that it thinks it needs to in order to complete the work that it has, with no intention of trying to get breaks along the way. This is typically the most common technique used in server power management today due to the system constraints preventing deep idle power savings.

CPU Power and Performance States

There exist a number of standard techniques for turning logic off as well as lowering the operating voltage inside of the CPU. This section will provide an overview of the power management capabilities that exist in the CPU and then go into detail about how each of the states performs under different environments. Table 2-11 provides an overview of the different power management states that are covered in detail in the following pages.

Table 2-11. *Overview of CPU Power Management States*

State	Granularity	Description
C-states	Core/thread	**Turning cores off and halting execution of instructions:** These states save power by stopping execution on the core. Different levels of C-state exist with varied amounts of power savings and exit latency costs. C1 is the state with the shortest exit latency but least power savings. Larger numbers, like C6, imply deeper power savings and longer exit latencies.
Package C-states	Package	**Turning off a subset of the package to save power when it is idle:** Package C-states kick in when all cores are in a C-state other than C0 (active). Like with core C-states, there can be multiple levels of package C-states that provide tradeoffs between power savings and exit latency. The package includes all the cores as well as other package blocks, such as shared caches, integrated PCIe, memory controllers, and so on. On Intel Xeon CPUs, these states typically have exit latencies <40 µs in order to avoid network packet drops.
P-states	Various	**Changing the frequency and voltage of a subset of the system:** Traditionally these states have been focused on the cores, but changing the frequencies of other components of the CPU is also possible (such as a shared L3 cache). Execution can continue at varied performance and power levels when using P-states.
T-states	Core	**Duty cycling the cores at a fixed interval:** T-states duty cycle the core execution to save additional power. These states are generally used for aggressive throttling when needed for thermal, electrical, or power reasons. They traditionally have not been used for power efficiency.

(*continued*)

www.dbooks.org

Table 2-11. (*continued*)

State	Granularity	Description
S-states	Package	**Turning off the entire package (sleep state):** These states are most common in client and workstation usage models, but can also be applied in some server CPUs. They tend to have very long exit latencies (seconds) but can drive the power close to zero. S0 represents the active state and S5 the "off" state (with multiples states in between).
G-states	Platform	**Global states:** These states refer to the power state of the platform. These are similar to S-states. G-states are generally not visible to the end user and are used by platform designers.
D-states	Device	**Devices (PCIe, SATA, etc.) in a powered-down state:** D-states are traditionally for devices such as PCIe cards and SATA and refer to low-power states where the device is powered down. D-states are not a focus on servers.

C-States

C-states provide software with the ability to request that the CPU enters a low-power state by turning off cores or other pieces of logic. A single CPU core may support multiple software threads if it supports simultaneous multithreading (SMT). Each HW thread has its own state and is given the opportunity to request different C-states. These are referred to as *thread C-states*, and are denoted as TC*x* (where *x* is an integer). In order for a core to enter a *core C-state* (denoted as CC*x*), each thread on that core must request that state or deeper. For example, on a core that supports two threads, if either thread is in TC0, then the core must be in CC0. If one thread is in TC3 and the other is in TC6, then the core will be allowed to enter CC3. Thread C-states themselves save minimal, if any, power by themselves, whereas core C-states can save significant power.

There are also *package C-states*, which can be entered when all the cores on that package enter into a deep core state. These states are commonly denoted as PC*x* or PkgC*x* (where *x* is an integer). At times, package state numbering is correlated to the state of the cores on that package, but this is not a hard rule. For example, the PC2 state on certain modern server processors is used when all cores on that package are in CC3 or CC6 states but other constraints are preventing the system from entering into a state deeper than PC2.

Thread C-States

Software requests C-states on a thread granularity. Minimal, if any, power savings actions are taking when a thread enters into a thread C-state without also inducing a core C-state. On CPUs that support SMT, these states are effectively a stepping stone to getting into core C-states. On CPUs that do not support SMT, thread and core C-states are effectively identical.

Core C-States

Core C-states determine if a core is on or off. Under normal execution, a core is said to be in the C0 state. When software (typically the OS) indicates that a logical processor should go idle, it will enter into a C-state. Various wake events are possible that trigger the core to begin executing code again (interrupts and timers are common examples).

Software provides hints to the CPU about what state it should go into (see Chapter 6 for more details). The MWAIT instruction, which tells the CPU to enter a C-state, includes parameters about what state is desired. The CPU power management subsystem, however, is allowed to perform whatever state it deems is optimal (this is referred to as *C-state demotion*).

Table 2-12 shows the C-state definitions from the Sandy Bridge, Ivy Bridge, and Haswell CPUs. There are no hard rules about how these states are named, but with a product line across generations, these definitions exhibit minimal changes.

Table 2-12. *Core C-State Examples*

Core C-State	Wake Latency	Description
CC0	N/A	**The active state (code executing):** At least one thread is actively executing in this state. Autonomous clock gating is common for unused logic blocks.
CC1	~1 µs	**Core clock gated:** In CC1, the core clocks are (mostly) gated. Some clocks may still be active (for example, to service external snoops), but dynamic power is driven close to zero. Core caches and TLBs are maintained, coherent, and available.
CC1e	~1 µs + frequency transition	**Enhanced C1—hint to drop voltage:** CC1e is effectively the same as C1, except it provides a hint to the global voltage/frequency control that V/f can be reduced to save additional power.

(*continued*)

www.dbooks.org

Table 2-12. (*continued*)

Core C-State	Wake Latency	Description
CC3	~50–100 µs	**Clocks gated and request for retention voltage:** Processor state is maintained, but voltage is allowed to drop to Vret. L1 + L2 (core) caches are flushed. Core TLBs are flushed.
CC6	~50–100 µs	**Power gating:** The core is power gated (voltage at 0). L1 + L2 (core) caches are flushed. Core TLBs are flushed. Processor state is saved outside the core (and restored on a wake).
CC7–CC10	Various	**CC6 with extra savings outside the core:** Additional states deeper than CC6 exist on certain CPUs. These states are generally not supported on server processors today due to their long latencies.

Core C0

Core C0 (CC0) is the active state when cores are executing one or more threads. The core's caches are all available. Autonomous power savings actions, such as clock gating, are possible and common. For example, it may be possible to clock gate floating point logic if integer code is being executed.

Core C1 and C1e

Core C1 (CC1) is the core sleep state with the fastest exit latency. Clock gating is performed on a large portion of the logic, but all of the core state is maintained (caches, TLBs, etc.). Some logic is typically still active to support snooping of the core caches to maintain coherency. Core C1 is a state that the core can enter and exit without interacting with the PCU. This enables fast transitions, but also prevents the global power management algorithms from taking advantage of this state for some optimizations.

Core C1e is a similar state, except that it provides a hint that the core can be reduced to a lower voltage/frequency as well. Although the exit from C1 and C1e are both about the same latency, it does take some time to ramp the core back to the requested frequency after the wake. The C1e state is generally achieved at the package granularity. In other words, all cores on a socket must first enter a C1e or deeper state prior to dropping the voltage/frequency on any core requesting C1e.

Core C3

Core C3 (CC3) provides gated clocks and a request to drop the voltage to retention voltage. It is conceptually a lower voltage version of C1e that does not require a frequency transition. C3 does, however, flush the core caches and core TLBs. It also has a much

longer wakeup latency than C1e. CC3 entrance and exits are coordinated with the PCU, so additional optimizations can take further advantage of this state (more later).

Core C6

Core C6 (CC6) saves a large amount of power by power gating the core. This requires the core to flush its state out including its caches and TLBs. Core C6 has a longer wakeup latency than CC3 (generally twice as long) because it must relock the PLL and ungate the power, but it can also save significantly more power than CC3 or C1e (the exact amounts vary significantly from product to product). This is the deepest possible power saving state for the core itself.

■ **Note** CC6 is the workhorse on servers for major idle power savings. CC1 is useful for saving power during short idle periods, or on systems where the latency requirements preclude the use of CC6. The CC3 state has generally shown minimal value in practice in servers. The performance impact of this state is similar to that of CC6 because of the cache flush, and dropping the voltage to Vret only occurs when all cores in the voltage domain agree to do so.

Core C7 (and up)

States deeper than CC6 are productized on many client devices. The core itself does not have any states deeper than power gating and CC6, but these deeper states can be requested by software and they provide a hint to the global power management algorithms about the potential for package-scoped power management optimizations (like flushing a shared L3 cache).

■ **Note** States deeper than CC6 have generally been challenged on servers,[6] because the server software environments rarely become completely idle. Flushing the L3 cache, for example, has non-trivial memory energy cost (both on entry and wake), and also results in longer wake periods on short wake events (because all data/code must be fetched from memory). These additional power costs tend to significantly offset (or even exceed) the power savings allowed by flushing the cache. Servers also tend to have much lower levels of latency tolerance, making further optimizations challenging.

---

[6]There was some confusion on the Sandy Bridge and Ivy Bridge generations because the CC7 state was enumerated in CPUID to software on Sandy Bridge, and then removed on Ivy Bridge. The CC7 state on Sandy Bridge E5 had identical power savings characteristics to CC6. As a result, to avoid long-term confusion, the CC7 state was removed on Ivy Bridge and does not exist on Haswell E5 or Avoton.

www.dbooks.org

C-State Demotion

The PCU can demote C-state requests made by software and decide to enter into more shallow states if it believes that the OS is asking for states that are sub-optimal. Early versions of software C-state control at times made overly aggressive requests for C-states when they were enabled, exposing some customers to performance degradation with C-states. In an attempt to resolve these concerns, C-state demotion was added into the PCU firmware in an attempt to prevent entry into deep C-states when it was determined by the processor that it could be detrimental to either performance or power efficiency. The details of these algorithms are not disclosed, and different algorithms have been deployed on different product generations. Although the PCU has worked to reduce the exposure to C-state performance degradation, operating systems have also tuned their selection algorithms to reduce their own exposure to performance degradation.

Early implementations of core C-state required OS software to save and restore both the time stamp counter (TSC) and local APIC timers. Recent processors have removed this requirement, and most of the work for entering a C-state and waking back up is handled autonomously by the CPU hardware and firmware.

Package C-States

When an entire CPU is idle, it can be placed into a package C-state in order to save additional power beyond what is possible with the subcomponents individually. These states are targeted at idle (or close to idle) conditions. The exact definition of these package states (what is turned off, and what the requirements are to do so) changes from CPU to CPU and generation to generation. However, the high-level concept remains the same.

When a CPU enters a deep package C-state, memory is no longer available to devices connected to the CPU (such as the network card). Intel servers commonly target a worst case of about 40 microseconds in order to restore the path to main memory for PCIe devices.

Table 2-13 provides an example of the package C-state definitions that are used across the Sandy Bridge, Ivy Bridge, and Haswell Server generations. Avoton did not implement package C-states and was able to achieve very low idle power without the need for a separate state managed by the power control unit. Instead, the power savings optimizations for idle power were implemented autonomously in the various IPs throughout the SoC.

Table 2-13. *Package C-State Examples*

Package C-State	Core C-States	Path to Memory	Description
PC0	At least one in CC0.	Available	The active state (code executing). No package-scoped power savings.
PC1e	None in CC0/CC1. At least one in CC1e.	Available	All cores have entered C1e or deeper states, allowing the opportunity for the voltage and frequency to drop. At least one core is still in C1e, preventing more aggressive power savings.
PC2	All cores in CC3/CC6.	Available	All cores are in CC3/CC6, but PCIe or a remote socket is still active. The shared uncore must still be active to support these other traffic sources. Minimal package-scoped optimizations can be performed here. The actions in this state are effectively identical to PC1e.
PC3	All cores in CC3/CC6. At least one in CC3.	Not available	All cores are in CC3/CC6 and other traffic sources (PCIe and remote sockets) are also idle. Package scoped operations, such as deep memory self-refresh or uncore Vret are possible.
PC6	All cores in CC6.	Not available	Same as PC3, except no cores are in CC3. Additional more aggressive power savings may be possible. On Ivy Bridge EP, for example, the L3 cache was only taken to retention voltage in PC6 and not in PC3.
PC7	All cores in CC7.	Not available	Same as PC6, except the L3 cache is also flushed.

■ **Note** The PC7 state has not been productized in many server processors (though it has been evaluated). Flushing the L3 cache costs memory energy and also causes any short-term core wakeups to take significantly longer, as all data/code must be fetched from memory. These added costs tend to significantly reduce the power savings that can be achieved with such a state, while also leaving the user with a longer wakeup latency.

www.dbooks.org

Module C-States

A *module* refers to a collection of cores that share resources. On Intel Atom-based server processors such as Avoton, groups of two cores share a single L2 cache. Other groupings are theoretically possible, such as sharing a single voltage/frequency domain. C-states are also possible at the module level and are commonly referred to as MC*x* (where *x* is an integer).

■ **Note** Module C-states have not been used as aggressively as core and package C-states in production on servers due to challenges in finding energy-efficient optimizations with them in server environments. These issues are similar to those observed with the flushed L3 cache in package C-states.

P-States

P-states were invented in order to dynamically reduce (or increase) the CPU operating voltage and frequency to match the needs of the user at a given point in time. Running at lower frequencies results in lower performance and longer latency to complete the same amount of work. However, it may be possible to complete a required amount of work with lower energy. A good example is a web server running a news web site. At 3:00 a.m., it is unlikely that many people will be accessing the data on that webserver. By running at a lower voltage/frequency, power can be saved. Each web request transaction on that CPU will take longer to complete, but in many cases the latency delta is so small relative to network transfer latencies that the customer will never notice. As the system load begins to increase, the frequency can be increased to meet the higher level of demand while a continued quality of service is maintained. The operating system has traditionally been responsible for selecting which frequency the system should operate at. See Chapter 6 for more details.

■ **Note** As shown in Figure 2-8, the voltage savings from decreasing frequency shrinks at lower frequencies (and eventually becomes zero). Decreasing frequencies past the point of voltage scaling is possible, but it tends to be inefficient. Users are better off using C-states at this point to save power. As a result, processors have a minimum supported operating frequency (called Pn) and may not expose lower frequencies to the operating system or allow lower frequencies to be requested.

P-states have since been extended to also transition voltage/frequency on other domains in order to save additional power. In some modern servers, the L3 cache and on-chip interconnect contribute non-trivial power to the CPU, and it is desirable to reduce the V/f of this domain when high performance is not required.

P-states are managed as a *ratio* of a *base clock frequency* (bclk). On the Nehalem generation, the bclk ran at 133 MHz. If the OS requested a ratio of 20, then the system would run at 2.66 GHz. All Xeon processors starting with Sandy Bridge have used a 100 MHz bclk. The Avoton architecture had a variable bclk that was based on the memory

frequency of the system. Each different ratio is commonly referred to as a *bin* of frequency. It is not possible to control frequency at granularity smaller than the bclk speed.

Voltage regulators (VRs) supply voltage to the CPU from the external platform (see Chapter 4 for more details). Having a large number of VRs to supply different voltages is expensive and challenging to manage/design. It is generally not power efficient to reduce the frequency of a system without also reducing the voltage. As a result, CPUs have supported a single variable voltage/frequency domain for the cores.

■ **Note** Having different cores on a CPU running at different frequencies but at the same voltage is suboptimal because frequency scaling without voltage scaling tends to be inefficient. As a result, most processors that are constrained to a single voltage domain for the cores are designed to require those cores to all run at the same frequency at all times.

In Haswell, Intel introduced the *Integrated Voltage Regulator* (IVR). This enables individual cores to have their own voltage (and therefore frequency) domains, enabling efficient per core P-states (PCPS). *Low-dropout regulators* (LDOs) can also be used to provide variable voltages across cores in a CPU with a single input voltage, but such a technique has not been productized by Intel to date.

Table 2-14 illustrates the progression of P-states in recent generations. Changes and innovation often occur on processors when a new platform is introduced since these optimizations have a platform design impact.

Table 2-14. *P-State Developments across Server Generations*

Generation	Base Clock	Core P-States	Uncore P-States	Comments
Nehalem Westmere	133 MHz	One variable domain	Static frequency (based on SKU)	
Sandy Bridge Ivy Bridge	100 MHz	One variable domain	Same voltage/ frequency as core domain	Uncore V/f scaling provides significant power savings at low utilizations.
Avoton	Variable based on memory speed	One variable domain	Static frequency (based on memory speed)	Uncore domain has low power contribution (no L3 cache).
Haswell E3	100 MHz	One variable domain	One dedicated variable domain	IVR used for separate uncore domain.
Haswell E5/E7	100 MHz	Per core variable domains	One dedicated variable domain	IVR allows per core control.

www.dbooks.org

Per Socket P-States

Certain processors such as Sandy Bridge, Ivy Bridge, and Avoton provide a single voltage/frequency domain across all cores on a socket. The target frequency is selected by looking across the requested frequencies on each of the threads with *voting rights* and taking the max of those frequencies. Voting rights are determined by the state that the thread is in, and that varies across generations. For example, a thread that is in a TC6 state may relinquish its voting rights on certain processor generations. It is important to note that the target frequency is not always granted—other aspects of the systems, such as temperature and power, may limit how high the frequency is able to go.

On Sandy Bridge and Ivy Bridge, voting rights were lost by any threads in C1e/C3/C6 states. This had two effects on the system. First, when all threads went into one of these C-states on a socket, no core on that socket would have voting rights and the core frequency would drop to the minimum frequency. Secondly, if different cores were requesting different frequencies, and a core requesting the highest frequency went to sleep, it could result in a decrease in frequency to the next highest requested frequency.

Avoton used a different approach. All cores maintained voting rights even when they were in C1e/C6 states (there was no C3 state on Avoton). However, a package C1e state was also used, which detected certain conditions when all threads were in a C1 or deeper state and would decrease the frequency to an efficient level.

Per Core P-States

Haswell E5/E7 provides the ability to independently change the frequency and voltage of the individual cores in the CPU.[7] In this mode of operation, the target frequency of a given core is simply the max of the requested frequency for the threads on the core. There is no concept of voting rights here.

Many servers execute workloads (like web servers) that service small, discrete "transactions." As the transactions come into the system, they are forked out to the various threads that service them. In this type of workload, different hardware cores tend to observe imbalances in utilization. These imbalances are constantly shifting and moving, but it is possible to take advantage of transient imbalances and reduce the frequency on cores that are underutilized. Figure 2-9 provides an example of one such workload. It compares management of P-states at the socket granularity (per socket P-states, or PSPS) to the per core granularity (PCPS). On the x-axis is the system utilization (with increasing utilization from left to right), and on the y-axis is the CPU socket power.

[7]This capability is not available on HSW E3 products.

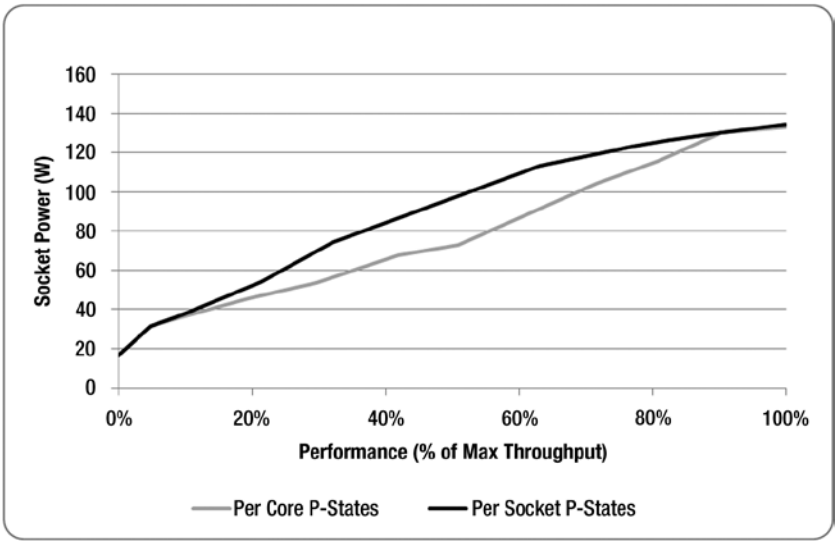


Figure 2-9. *Per core P-states (PCPS) vs. per socket P-states (PSPS)*

When running different workloads on a system, it can be useful to execute them at different frequencies. A common example of this is with virtualization. One user may desire 100% of their virtualized resources, whereas another may be running at very low utilization. PCPS allows the active user to ramp their voltage and frequency up without imposing those power costs on the second user. A similar situation exists with different types of workloads running on a system. If a subset of cores is being used for some performance-critical task and are running at a high frequency while another core periodically wakes up to service a daemon, there is no need to execute that daemon at the high voltage/frequency point. When threads wake up and execute at a high voltage/frequency, they theoretically can get into a deep C-state faster (mitigating the cost of the high frequency, or even turning it into a net power savings). It is not uncommon for server platforms to have a significant set of background software threads that can perturb the system, it can cause threads to wake up frequently, thus preventing the use of these deep C-states at moderate to high system utilizations. This behavior is much less common on power-optimized consumer platforms where it would cause significant battery life degradation (with or without PCPS).

Per core P-states are not always a huge win. A good example of this is with low-power microservers. In microservers, it is common for the amount of power consumed by the IA cores to be a smaller percentage of the overall platform power. It is also common for these CPUs to run at lower frequencies with smaller voltage dynamic range. Without good voltage scaling, you are better off racing to halt on an individual core and getting into a deep C-state on that core rather than reducing the frequency and voltage of that core alone.

www.dbooks.org

Uncore Frequency Scaling

The Nehalem and Westmere families of processors maintained a constant frequency in the uncore. At low system utilizations (about 10%–40%), this was an inefficient operating condition, because the L3 cache was kept at a higher frequency and voltage than necessary. However, it provided generally consistent performance behavior.

On the Sandy Bridge and Ivy Bridge generations, the uncore and cores on a socket were tied together into a single voltage/frequency domain. When the cores changed frequency, the uncore (L3 cache) moved with them. This provided significantly better power efficiency at low system utilizations, since the L3 cache voltage was reduced, saving leakage power. In addition to this, many server workloads saw improved frequency scaling efficiency (larger performance increases by increasing frequency).

On Haswell, the cores and uncore were moved to separate variable voltage/frequency domains. This allows the system to take advantage of all the benefits of a variable uncore domain, while also allowing for improved power efficiency. For example, if one socket in a two-socket system desires high performance and the second socket is idle, it informs the second socket that it is in a high-performance mode. The idle socket is then able to increase the frequency of its uncore in order to supply the best memory and snoop latencies to the high-performance socket without increasing the voltage/frequency of the idle cores on that idle socket. This feature is called *perf p-limit*. Similar behavior is possible when high performance is required by PCIe.

Avoton does not have an L3 cache or a high-power uncore like is commonly found on Xeon processors. As a result, managing the uncore frequency is simply not worth the cost in that case.

Turbo

CPU server platforms are typically designed to provide sufficient cooling for relatively worst-case real workloads and power delivery capabilities. In servers, the vast majority of the workloads that are typically run on these systems run well below these constraints that they are designed for. Turbo was introduced to take advantage of this dynamic headroom. It increases the operating frequency of the CPU in order to take advantage of any headroom for

- Power
- Thermals
- Electricals

In order to provide additional frequency beyond the base frequency of the unit, headroom must exist in each of these three major areas. The amount of Turbo that can be achieved is dependent on the thermals of the platform/data center, the workload being run, and even the specific unit.

Turbo Architecture

The Turbo architecture/micro-architecture is largely shared across the Intel product lines (from phones/tablets up to E7 servers). However, the behavior of these algorithms has generally been different in each domain. In consumer devices (laptops, tablets, etc.), it is not uncommon for users to require short-term performance boosts. These platforms are also frequently thermally constrained. Turbo provides additional performance while the temperature increases (both internal to the CPU as well as on the device "skin" that people touch). With some workloads, the thermal capacity will eventually run out, and the CPU must throttle back its frequency in order to stay within the thermal constraints of the platform. The Turbo architecture introduced in the Sandy Bridge generation (called Turbo 2.0 or Running Average Power Limit [RAPL]) attempted to model these thermal characteristics and provide a mechanism for staying within a desired thermal constraint, both in the actual CPU as well as at the platform. On servers, it is not uncommon for certain workloads to sustain high levels of Turbo frequency indefinitely.

Power/Thermal Limits

Thermal constraints generally track directly with power usage over long-time constants. A laptop, for example, can dissipate a certain amount of power/heat without changing temperature. Use more power, and the laptop will heat up; use less, and it will cool down. The Turbo algorithms model these behaviors and constrain power over thermally significant time constants (usually seconds) in order to stay within the desired thermal envelope. These same algorithms exist in servers and work to keep the CPU within a desired power/thermal envelope. See the section "T-States" for more details.

Thermal Protection

In addition to controlling thermals through power limiting, the CPU provides thermal management routines that keep the CPU operating within its thermal specifications. These thermal algorithms are enforced during Turbo as well. They are documented in "CPU Thermal Management" section.

Electrical Protection

Although both power and thermals can generally be dealt with reactively, electrical constraints are generally less forgiving. The power delivery of the platform has a maximum current that it can supply (called ICCMAX). This limit typically comes from the voltage regulators (both IVR and MBVR), but CPU package and socket constraints are also involved. Exceeding the ICCMAX of a VR for shorts periods of time (microseconds) can result in a voltage droop and a system failure. These time constants are too fast to detect and react to reliably today, and as a result, a combination of *proactive* enforcement and platform design constraints must be used to prevent system failure. The Turbo algorithm has an electrical design point (EDP) limit that detects when it may be possible to exceed the ICCMAX of the processor and reduces frequency proactively to avoid these problems. Typical workloads will see little to no EDP throttling, because the CPUs are tested to ensure that it is possible to *electrically* achieve maximum Turbo under most conditions.

www.dbooks.org

The big exception to this rule is with advanced vector extensions (AVX) workloads. AVX is a set of wide-vector instructions targeted primarily at high-performance computing and other math-heavy applications. These instructions have the potential to consume significant power and pull significant current. As a result, when AVX instructions are in use, the EDP algorithm can push the frequency down by one or more frequency bins. AVX can significantly improve both performance and power/performance efficiency, but it can also reduce overall performance if only lightly used.

■ **Note** AVX has the potential to consume significant power when used. However, when it is not in use, much of the logic can be automatically (and dynamically) gated off, and the CPU does not need to take AVX into account for electrical protection calculations. There are generally no BIOS knobs or OS knobs to disable AVX, since it has minimal cost to workloads that do not make use of it.

Table 2-15 illustrates the behavior of EDP across generations. In Sandy Bridge, EDP did not exhibit a significant impact on system behavior. On Ivy Bridge, EDP throttling was more common. This throttling was applied across the entire socket. In other words, if one core was using AVX, all cores were throttled to stay within the limits. Haswell operates in a manner similar to Ivy Bridge. However, separate constraints were included that defined the level of Turbo that was possible when AVX was active.

Table 2-15. *Turbo Electrical Protection Across Generations*

Generation	EDP Throttling
Sandy Bridge E5	Not common. Applied per socket.
Ivy Bridge E5	Common with AVX. Applied per socket.
Haswell E5	Common with AVX. Applied per socket. Hard limits on Turbo applied for AVX codes.[8]
Avoton	None.

C-States and Turbo

C-states not only save power but also can provide additional performance when used with Turbo. By placing cores into deep C-states (C3 or deeper), it can be possible to grant higher Turbo frequencies. Not only do C-states save power that can be spent on Turbo, but, when in a sleep state, the PCU knows that the cores cannot suddenly require high current. This means that the platform ICCMAX constraints are divided up across fewer cores, allowing them to achieve higher frequencies. This can be particularly useful in workloads that have a mix of parallel and serial portions, because the serial portions

[8]See www.intel.com/content/dam/www/public/us/en/documents/white-papers/performance-xeon-e5-v3-advanced-vector-extensions-paper.pdf.

can achieve higher frequencies and complete faster. The core C1e and C1 states are not negotiated with the global PCU in order to provide fast wake and sleep responsiveness. They also do not reduce the voltage, and some hardware continues to operate (such as the path to snoop the caches that are not flushed). As a result, use of the C1 and C1e states can slightly improve Turbo performance by saving power, but additional Turbo bins are not made available.

■ **Note** C-states commonly increase peak performance of certain workloads when used in conjunction with Turbo by allowing higher frequencies to be achieved when the number of active software threads is less than the number of available hardware threads.

Fused Turbo Frequencies

Each processor SKU is fused with a base (P1) frequency as well as a max Turbo (P0) frequency. In addition to these two points, limits are commonly imposed on Turbo depending on the number of active cores. For example, an 8-core CPU may have base frequency of 2.8 GHz and a maximum Turbo frequency of 3.6 GHz, but it may only be allowed to achieve a frequency of 3.2 GHz if all of the cores are active, or a frequency of 3.4 GHz if four cores are active. Many server workloads make use of all available cores while running with Turbo and are therefore limited to the all-core Turbo frequency (P0n frequency). The supported maximum Turbo frequencies for different numbers of active cores are referred to as the *Turbo schedule*. On Haswell, the Turbo schedule concept was extended to AVX. In addition to the legacy Turbo schedule, an additional set of fused limits was added and applied when AVX workloads are active.

T-States

T-states provide a mechanism to duty-cycle[9] the core in order to achieve even lower levels of power savings than are possible with P-states without depending on the operating system to request a C-state with MWAIT. T-states are a very inefficient way to save power and are generally used exclusively in catastrophic situations to avoid system shutdown or crash. T-states can be requested by the operating system or entered autonomously by the CPU when it detects severe thermal or power constraints. Modern operating systems do not make use of the T-state request infrastructure, but it is maintained for legacy purposes.

T-states are generally implemented using course-grained duty cycling between a C1-like state and C0 state (10s to 100s of microseconds of clocks being gated, followed by a period of being active). However, it is also possible to use fine-grained clock modulation (or clock duty cycling) to implement these states, or course-grained duty cycle with deeper C-states.

---

[9]T-states technically also include frequency reduction below the point where voltage reduction is possible.

www.dbooks.org

S-States and G-States

S-states and G-states provide deep power management at the platform level. S-states are software (and end-user) visible, while G-states are targeted primarily at platform designers. Software must request for a CPU to enter into an S-state, and a wakeup from an S-state requires software (BIOS and OS) support. This is different from package states where the wake is managed entirely by the CPU. The S0/S4/S5 states are supported by most server CPUs. S3 is generally more of a workstation and client feature and is not supported by all server processors. Table 2-16 provides a summary of some of the common S- and G-states.

Table 2-16. *S-States and G-States*

G-State	S-State	Description
G0	S0	The CPU is powered on and managing its own power.
G1 (sleeping)	S1/S2	Legacy sleep states that have since been replaced by package C-states.
	S3 (suspend)	CPU (mostly) turned off with state saved in DRAM for fast wake (seconds).
	S4 (hibernate)	CPU completely turned off with state maintained on drive for improved wakeup latency.
G2 (soft off)	S5 (soft off)	CPU is completely turned off with no state saved. Some minimal power still provided by the PSU to enable wakeups (button press, keyboard, WoL [Wake on LAN], etc.). Wake from this state can take many seconds to minutes.
S3 (mechanical off)	N/A	PSU is no longer providing any power. Some minimal power may still exist for maintaining the system clock or minimal otherwise volatile states.

S0ix

S0i*x* states provide power savings that are conceptually similar to package C-states. They provide global optimizations to save large amounts of power at an idle state. There are varied levels of S0i*x* (today from S0i1 to S0i3) that provide successively deeper levels of power savings with increasing exit latencies. The S0i*x* terminology has predominantly been used in consumer devices and not in servers. The exact definition of these different states has (to date) changed from generation to generation.

Running Average Power Limit (RAPL)

Imagine having a car that had a top speed of 35 mph, and whenever you tried to drive the car faster than 35 mph, it would react by dropping the speed down to 32 mph. In such a situation, it would be very difficult to sustain 35 mph. This is conceptually how Turbo behaved on the Nehalem generation of processors. Whenever power exceeded the allowed threshold, the frequency would be decreased in order to get back below the limit. Frequency was managed on 133 MHz increments with only about 10 different options for which frequency could be selected (imagine a gas pedal that had only 10 different "options" for how hard you could press), causing the system to drop below the target max power level. As a result, in workloads that were power constrained, it would be difficult to make use of the full capabilities of the system.

Sandy Bridge introduced the concept of *Running Average Power Limit* (RAPL) for controlling power usage on a platform to an average limit. RAPL is a closed loop control algorithm that monitors power and controls frequency (and voltage). On prior generations, the Turbo algorithm attempted to keep the power below a limit. Whenever power exceeded that limit, frequency would be reduced in order to get it back under the limit as quickly as possible. With RAPL, exceeding the power limit for short periods of time (usually up to a few seconds) is okay. The goal of RAPL is to provide an average power at the desired limit in a manner that will keep the system within the thermal/power constraints.

Platforms have a variety of different constraints that must be met in order to keep the system stable. There are a variety of different thermal requirements (e.g., not over-heating the CPU, VRs, PSU, memory, and other devices) as well as power delivery requirements (e.g., staying under the ICCMAX of the VR). Many of these constraints will be discussed in Chapter 4. RAPL provides capabilities for addressing a number (but not all of) of these different constraints.

Different components/constraints in the platform have different power requirements. Some constraints are loose—they can be broken for certain periods of time. Others are hard constraints, and breaking them can lead to failures. Table 2-17 provides some examples of these constraints.

www.dbooks.org

Table 2-17. *Platform Power Constraints Example*

Platform Constraint	Typical Power Constraint	Notes
Voltage regulators	~2 times TDP power	Exceeding the constraints of the voltage regulator for short periods of time (microseconds) can lead to system failure. These limits are typically hard limits.
Power supply	~1.2 times TDP power	Power supplies and the platform can burst to higher power levels for periods of time (typically milliseconds). These time constants can be increased with additional cost.
CPU thermals	~TDP power	It typically takes time for the CPU to heat up. As a result, exceeding the thermal power budget for a short period of time can be acceptable (while the system heats up). These time constants are platform- and workloads-specific, and are typically in the hundreds of milliseconds to seconds. The CPU will protect itself if it detects that temperatures are exceeding the specified limits.

RAPL is targeted at controlling a number of (but not all of) these requirements. Different levels of RAPL provide protection for different time constants that are targeted at different platform constraints (see Table 2-18). These capabilities have evolved over time (see Table 2-19). RAPL provides one mechanism (PL1) for controlling average power over thermally significant time constants (seconds). The goal is to maximize the total power available while staying within the configured constraints. It also provides additional mechanism (PL2/PL3) for controlling the system over much shorter time constants in an attempt to stay within various power delivery constraints. These limits are typically higher than PL1 but must be enforced over much smaller windows of time. Unlike thermally constrained consumer platforms (like small form factor laptops), the exact PL2 and PL3 values are generally less critical to overall system performance, and typically are not aggressively tuned.

Table 2-18. *RAPL Levels*[10]

Level	Time Constant	Target Usage	Example Configuration
PL1	Seconds	Thermals + average power	TDP
PL2	~10 ms	Thermals + power delivery	~1.2 times TDP
PL3	<10 ms with duty cycle	Power delivery	~1.2 times TDP
ICCMAX	Proactive	Power delivery	SKU specific

[10]Values in this table are provided as typical examples. They are not in any way hard limits, and the values are all programmable by the system designer (within certain constraints).

Table 2-19. *RAPL Capabilities Across Product Generations*

Product	PL1/PL2	PL3	ICCMAX	Memory
Sandy Bridge/ Ivy Bridge	Supported	Not supported	Static, decided at boot	Per socket
Haswell	Supported	Supported[11]	Dynamic control	Per socket
Avoton	Supported	Not supported	Not supported	Not supported

Sandy Bridge implemented PL1 and PL2 time scales. By default, PL1 is set to the TDP power of the processor/SKU, and PL2 is set to about 1.2 times TDP. Each of these limits can be set statically (by BIOS) or controlled dynamically at runtime (through either PECI or IA software). Any limits for PL1 set above the TDP power level will be clipped to TDP (with the exception of high-end desktop processors that support overclocking). Although this worked well in some usage models, supporting only PL1 and PL2 made it difficult to use RAPL for power delivery protection. It was still deployed for data center power budgeting and control, but guard bands were required.

Haswell extended the capabilities on Sandy Bridge to attempt to better address short-term power delivery constraints. In addition to PL1/PL2, a third constraint (PL3) was added to the system that can detect power excursions on shorter time constants and throttle with deterministic duty cycles. This enabled less power delivery over-design (particularly at the granularity of the PSU).

Sandy Bridge also enforced an ICCMAX limit. As discussed previously, ICCMAX is enforced proactively so that it is never exceeded. On Haswell, ICCMAX became programmable at runtime. This allowed for the data center management software to set a hard limit on the max current/power that would never be exceeded.

Figure 2-10 provides an illustration of PL1 and PL2 in operation (not to scale). The PL3 power level conceptually operates in a similar manner as PL2, just with more well-defined behavior that is more amenable to platform design. The x-axis of both graphs represents time. As time goes from left to right, different workload phases execute (as shown by the “Activity” at the bottom of the chart). To start, the workload is in a low activity phase (such as memory allocation). Despite frequency running high, the actual power is low. In this phase, the temperature will generally be relatively lower, and the control loop can acquire these power credits to spend later.

[11]PL3 was supported on HSW E5/E7. On this processor, the power level was shared with the PL2 power level. On HSW E3 PL3 used a separate configurable power level from PL2.

www.dbooks.org

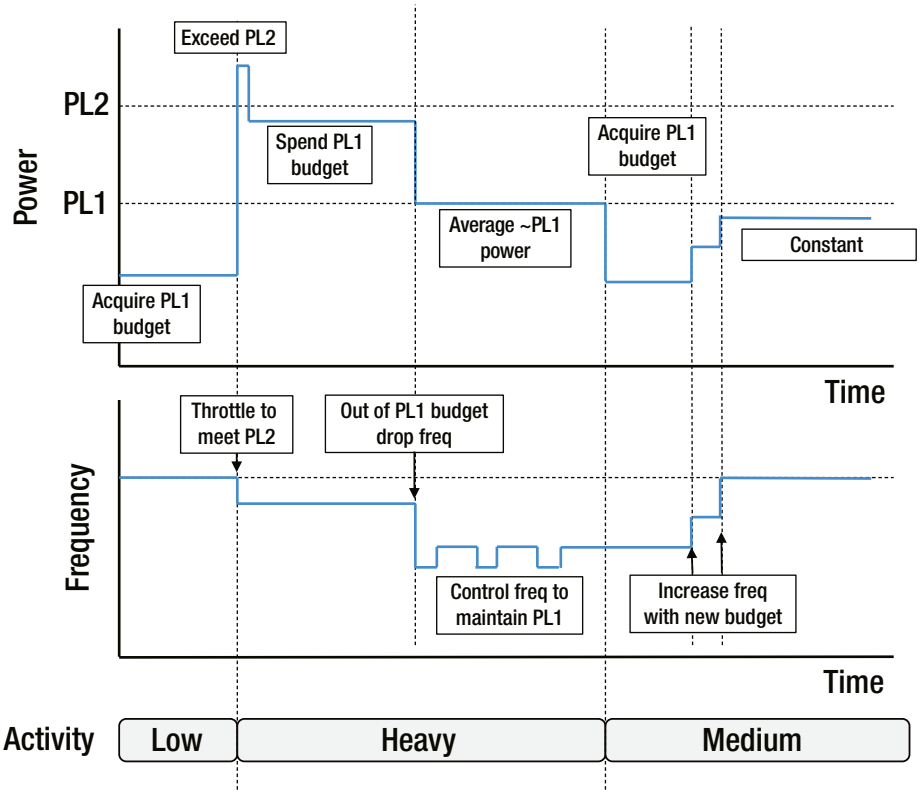


Figure 2-10. *Illustration of power-throttling with Turbo 2.0*

Then, the workload transitions into a "heavy" phase. At the high frequency, the heavy workload exceeds the PL2 level and is quickly throttled back down until it is below PL2. It is then able to sustain a slightly lower frequency for a while despite the average power being higher than PL1. The CPU is effectively spending the energy credits that were saved up while the power was low. This is intended to model the thermal capacity of the system. It is okay to run above the PL1 power for a while as the heat sink heats up. Once those credits are used up, the frequency will drop further in order to sustain the PL1 average. The PL1 control loop will periodically increase and decrease the frequency such that the running average matches the PL1 constraint.

Finally, the workload completes the heavy phase and transitions into a phase of medium activity. The power drops as the activity reduces. After a short period, the control loop acquires enough budget to begin increasing the frequency again. In this case, the frequency stabilizes at the maximum supported frequency as the power consumed at that level is below the PL1 constraint. This mode of operation is actually quite common on many server workloads that consume less than the PL1 power even at the max supported Turbo frequency.

The RAPL concept can be applied to any power domain that supports power reporting and a mechanism for providing throttling to control power. DRAM RAPL provides an interface to control power to the DRAM domain. PP0 RAPL existed on the Sandy Bridge and Ivy Bridge generations for controlling the power of the core power domain (VCC). This was not found to be particularly useful in production and therefore was removed in the Haswell E5 generation.

IMON and Digital Power Meter

In order to provide a closed-loop algorithm for RAPL, it is necessary to provide power-measurement feedback. There are two high-level ways to do this: (1) measure the power/current with an analog circuit, or (2) estimate the power using logic inside the CPU. Voltage regulator current monitoring (VR IMON) is the primary option for number 1. As the VRs supply current to the CPU, a circuit within the VR keeps track of an estimate of the power. The CPU then periodically (usually ~100 μs to ~1 ms) samples this reading and calculates power from it. The alternative to this is to use a *digital power meter* to implement number2.

VR IMON is generally significantly easier to implement/tune for the CPU but adds some platform cost. For a single VR, these costs are generally small (much less than $1). It does have the drawback that the VR circuit must be tuned for accuracy. The digital power meter provides a mechanism to estimate power without the platform requirement. Most server designs leverage VR IMON, because it provides good accuracy with lower effort. The exception here is the CPU on Sandy Bridge and Ivy Bridge, which used the digital power meter. VR IMON also typically includes some simplified digital power meter for a subset of the die. For example, on Avoton, there are a large number of input VRs. Many of those VRs supply a small and (generally) constant voltage/current to the CPU. Rather than implement VR IMON on these rails (increasing platform cost and design complexity), a simple digital power meter is used to estimate power for those rails. Table 2-20 illustrates how power monitoring has evolved over recent processor generations.

Table 2-20. *Turbo Power Monitoring/Enforcement Across Generations*

Generation	Throttler	Power Measurement
Nehalem/Westmere E5	Turbo 1.0	CPU: VR IMON DRAM: N/A (not supported)
Sandy Bridge/ Ivy Bridge E5	RAPL (Turbo 2.0)	CPU: digital power meter DRAM: VR IMON
Haswell E5	RAPL (Turbo 2.0)	CPU: VR IMON DRAM: VR IMON
Avoton	RAPL (Turbo 2.0)	CPU: VR IMON DRAM: VR IMON

www.dbooks.org

■ **Note** VR IMON is typically optimized at the max current level that the VR can supply. As load reduces, the amount of error is mostly constant (in amps). However, as a percentage of the load, the error increases. As an example, 1 A of error out of 100 A is only a 1% error. However, at a utilization of 10 A, this error becomes 10%. So, when systems are idle, both the DRAM and CPU IMON reporting tends to exhibit higher errors. Platform memory power (and current requirements) can vary significantly based on the amount of memory capacity. DRAM VR inaccuracy can be large (as a percentage) on systems allocated with much lower capacity than the platforms are capable of.

Linpack Example

Linpack (HPL) is a terrible workload for illustrating *typical* server workload power behavior. However, it is excellent at stressing a system and demonstrating the behavior of the RAPL algorithm and therefore is used here. Many typical server workloads that run with the default system configuration (PL1 = TDP) will not experience any throttling from RAPL and can sustain Turbo indefinitely.

Figure 2-11 shows the behavior of Linpack (HPL) over a subset of the workload run with RAPL engaged at a temperature of 85°C and a 1 s time constant. There are a couple of interesting observations from these data. First off, they illustrate the overall behavior of RAPL in a real workload. In the beginning (to the left), Linpack is performing memory allocation and consuming relatively low power despite the high frequency. Next, the actual workload kicks in. Power jumps up above the PL1 limit. After a number of seconds, the RAPL PL1 limit kicks in and brings the power down to the TDP/PL1 limit. At this time, frequency drops off by about 100 MHz in order to sustain the 85 W limit.

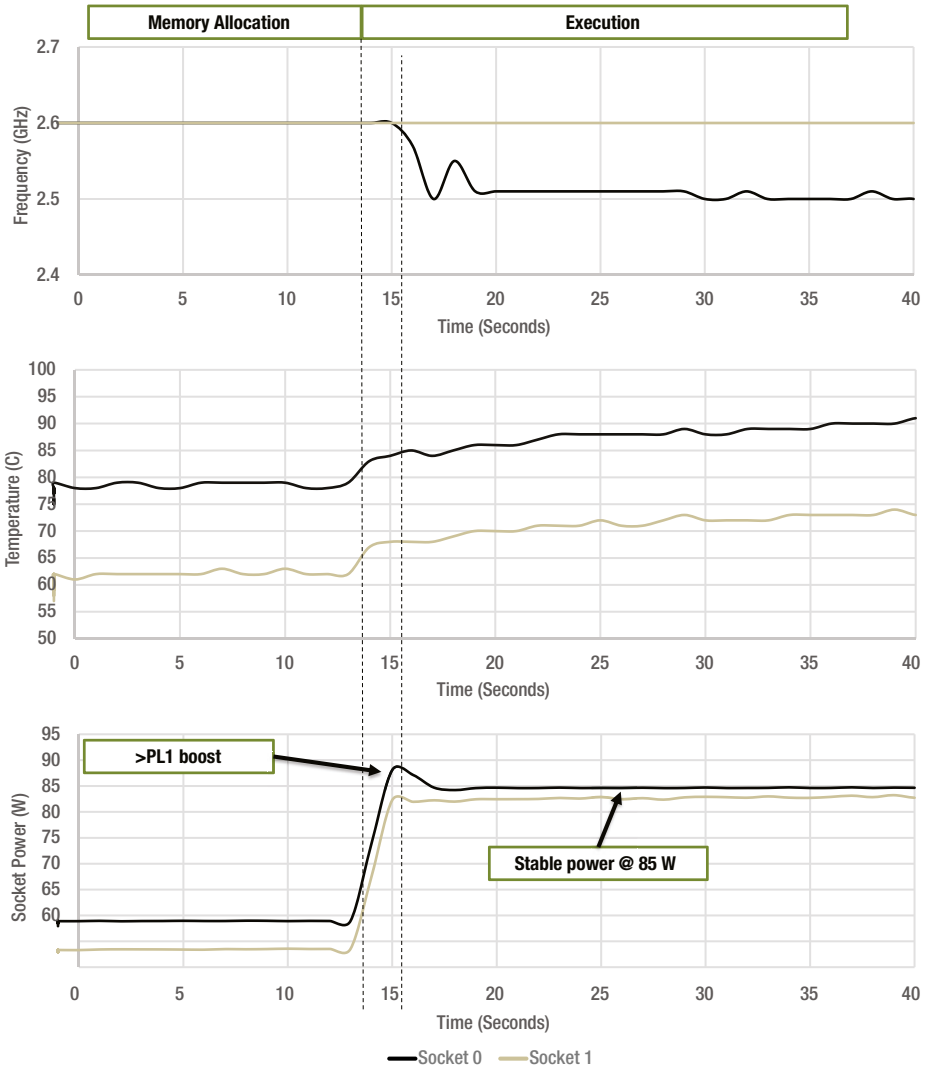


Figure 2-11. *Linpack power, frequency, and temperature with 85 W RAPL limit*

Second, Socket 0 consumes more power and achieves less frequency (and performance) than Socket 1. In the platform studied, Socket 0 is in the thermal shadow of the socket. In other words, the fans were blowing air first over Socket 1 and then that heated air passed over Socket 0. The result is that the temperature of Socket 0 is much

www.dbooks.org

warmer than Socket 1, increasing the leakage power consumed by that CPU. As a result, Socket 0 achieves lower average frequency at steady-state, and its initial boost when the workload starts executing lasts for less time.

Many typical server workloads will not show this type of performance variability across sockets as they tend to achieve the maximum supported Turbo frequency even at higher temperatures. However, when lower power limits are engaged, this sort of variability can be observed. Data center management utilities monitor the achieved performance levels across different sockets and different nodes in the data center in an attempt to balance out the necessary power to optimize performance.

DRAM (Memory) RAPL

In addition to the socket domain, the Xeon E5 line supports DRAM RAPL, which provides power limiting to the memory domain. Memory power can be a significant portion of the overall platform power. This was particularly the case with 1.5 V DDR3. With the transition to DDR4, this contribution has decreased but still remains important, particularly in large memory capacity systems.

DRAM RAPL is conceptually very similar to socket RAPL. Power is monitored over a time window, and throttling is performed in order to stay within a designed power limit. With CPU RAPL, power is modulated by controlling the voltage and frequency of the system. With DRAM, changes to voltage and frequency are not common. As a result, power is controlled by limiting the amount of transactions to the DRAM devices. DRAM power is very sensitive to bandwidth. Unlike socket RAPL, which supports separate PL1 and PL2 power levels and time constants, DRAM RAPL today only supports a single configuration point. There is also no ICCMAX control point, although proactive peak bandwidth control can be performed using the thermal management infrastructure.

The DRAM power domain is separate from the CPU domain. The two cannot automatically share power today. The VRs that power memory typically also supply power to the DDR I/Os that exist on the CPU. This power is included in the CPU domain (typically using some form of digital power meter). In order to avoid double-counting, this power is subtracted from the DRAM RAPL. Data center management software can implement algorithms that allow for power to be shared between the Socket and DRAM domains. Although the two domains are separate, they will interact with each other. Setting a strong CPU RAPL limit that results in heavy CPU throttling will generally result in lower DRAM power because the lower CPU performance will result in lower DRAM bandwidth. The side effects to CPU power caused by DRAM RAPL are less obvious. When DRAM RAPL throttling is engaged, the cores spend more time stalled. These stalls will reduce the activity of the cores and reduce their power (in the short term). If those cores were running at a low frequency, the OS may observe a higher utilization and increase the voltage and frequency, ultimately increasing the CPU power. On the other hand, if the frequency is already running at the max, the power will be decreased.

Throttling memory is generally a very power inefficient action. Cores are left stalled and unable to efficiently complete work in order to enter a C-state. As a result, under normal operation, DRAM RAPL is generally used to limit power at a level slightly higher than the needs of the workload. For example, if a workload is consuming 20 W unthrottled but the system could consume up to 30 W, a 20 W limit could be deployed that avoids throttling the workload but also prevents it from jumping up to 30 W.

The remaining 10 W can then be spent elsewhere by the management software. If power needs to be reduced and throttling needs to occur, it should generally start with the CPU domain and then only move to the DRAM domain as a last resort.

■ **Note** DRAM RAPL is most effectively used to ensure that you don't over-provision unnecessary power to DRAM. However, throttling memory should be avoided except when critically necessary.

CPU Thermal Management

Maintaining a safe operating temperature is critical to long-term functionality of a CPU. Managing the platform cooling to keep the CPU within an optimal temperature range is typically the responsibility of platform software and is discussed in detail in Chapter 4. However, the CPU itself monitors its own temperature and provides automatic thermal throttling mechanisms to protect the CPU from damage or data from being lost.

The CPU keeps track of the internal temperature (Tj or junction temperature) of the die using multiple thermal sensors. If these thermal sensors detect a temperature larger than the max allowed temperature of the SKU (DTSMAX), the operating frequency is throttled back to stay within the thermal constraints. Thermal throttling through this mechanism is generally not common, but it has been developed to provide good performance when in use. Frequency is generally throttled *slowly* while the temperature exceeds the desired levels. Thermals inside of a CPU do not respond instantaneously to changes in power/frequency due to non-trivial thermal resistance. Temperature does not typically change much faster than about every 10 ms (and commonly much slower). As a result, the thermal throttling algorithms are tuned to reduce frequency and evaluate its impact on temperature over millisecond time scales before further reduction in frequency is performed.

If the temperature begins to exceed the DTSMAX by a large amount, aggressive throttling (typically to the minimum supported frequency) is performed in order to quickly reduce temperature. This is commonly referred to as a *critical temperature event*. In servers, this occurrence is very uncommon, and typically only happens when there is a catastrophic issue with the cooling capabilities of the platform/rack (i.e., a fan or two stops working). When this type of throttling is engaged, the goal is to keep the system functional until the platform issue can be diagnosed and resolved. Performance is not a priority. It is possible to configure the OS/BIOS to attempt a "graceful" shutdown (from software) when this event occurs, but this capability is typically not enabled in server systems and the aggressive throttling is relied upon instead.

In addition to the DTSMAX, each unit is fused with a catastrophic trip temperature that is typically referred to as *THERMTRIP*. When the temperature exceeds this fused limit, the CPU immediately signals to the platform (through a pin) that an immediate hardware shutdown (without OS intervention) should be performed. This capability is implemented entirely in simple, dedicated asynchronous hardware, and is intended to function even if other failures occur within the CPU. In other words, the cores and internal microcontrollers could all hang, and the clock network could fail, but

www.dbooks.org

THERMTRIP would still be operational. It is very rare to observe THERMTRIP in production units, and it can even be difficult to induce it in the lab without disabling the other thermal control algorithms.

■ **Note** Thermal throttling can occur from improper cooling (e.g., a fan failure or a poor thermal design) or because of Turbo consuming all of the thermal headroom that is available. The thermal reporting mechanisms that exist on modern processors do not differentiate between these two cases, and this can lead to some confusion by end users.

Figure 2-12 provides an example of Linpack when it is being exposed to thermal throttling on Socket 0. Similar to the example in Figure 2-11, in this case Linpack is being run on a system where Socket 0 is in the thermal shadow of Socket 1, causing it to run at higher temperatures. At the beginning of the workload, memory allocation is performed and the system is able to run at the full 2.6 GHz frequency without significant heating. Once memory allocation is complete and the actual workload begins to run, power increases significantly and the processor begins to warm up. At about the 150-second mark, Socket 0 begins to hit the DTSMAX temperature of 95°C, and frequency begins to throttle in order to keep the CPU below the 95°C temperature. Frequency decreases until it stabilizes at an average frequency of ~2.45 GHz. Note that in this case, the CPU is actually switching between the 100MHz frequency bin granularities (2.4 GHz and 2.5 GHz, primarily), and it is the average frequency that sustains ~2.45 GHz.

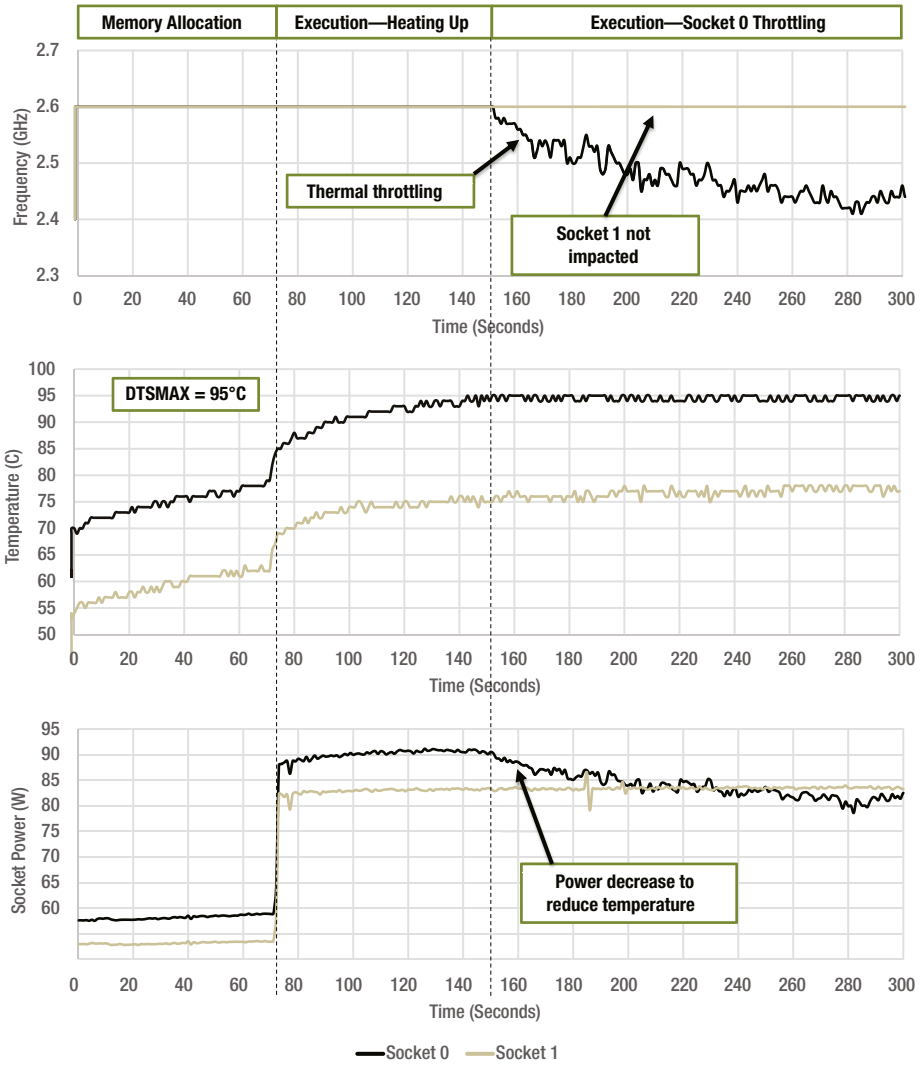


Figure 2-12. *Linpack frequency, temperature, and power when thermally throttled*

www.dbooks.org

■ **Note** In many versions of the Linux kernel, any sort of thermal throttling is commonly reported as a "concerning" error message. Thermal throttling, while uncommon in servers, is something that will frequently happen over the life of a product and there is nothing wrong with the system. When quickly transitioning from low-power workloads into very high-power workloads, the temperature of the CPU can frequently increase faster than the fan speed control algorithms can react to keep the temperature below DTSMAX. The CPU thermal management algorithms are in place to protect the CPU from damage and react gracefully to control frequency to stay within the thermal budget.

Prochot

There exists a pin called PROCHOT# on modern server CPUs that can both provide an indication of when the CPU is being thermally throttled (output mode) and be used as a mechanism for the platform to tell the CPU to throttle (input mode). It can also be used as a bidirectional pin so that both modes can be used simultaneously. Prochot output mode can be used for various platform usage models. The input indicates to the CPU that it should perform a heavy throttle as quickly as possible (usually to a low frequency). Haswell improved the speed of the Prochot mechanism so that it could be used for power delivery protection. More details are in Chapter 4.

CPU Power Management Summary

Figure 2-13 provides a high-level example of the various states that software and the CPU can employ to save power through a combination of "turning off" and "turning down."

		Turn It Off		Turn It Down		Resources			
	State	Clock Gating	Power Gating	Voltage	Frequency	Core Caches	Core TLB	Uncore Cache	DDR Memory
Core C-states	C0	Autonomous	-	-	-	Active	Active	-	-
	C1	Yes	-	-	-	Snoopable	Saved	-	-
	C1e	Yes	-	Requested down	Requested down	Snoopable	Saved	-	-
	C3	Yes	-	Requested Vret	Requested down	Flushed	Flushed	-	-
	C6	-	Yes	Off	Off	Flushed	Flushed	-	-
Package C-states	PC0	Autonomous	-	-	-	-	-	-	CKE, OSR
	PC1e	Autonomous	-	Reduced	Reduced	-	-	-	CKE, OSR
	PC6	Select logic	Select logic	Reduced	Reduced	Flushed	Flushed	V_RET	Self-refresh
	PC7	Select logic	Select logic	Reduced	Reduced	Flushed	Flushed	Flushed	Self-refresh
P-states	Turbo	-	-	High	High	-	-	-	-
	P1	-	-	Moderate	Moderate	-	-	-	-
	Pn	-	-	Min	Max @ min voltage	-	-	-	-
	Pm	-	-	Min	Min	-	-	-	-
S-states	S0	-	-	-	-	-			
	S3	-	-	Off (0v), except DDR	-	Off	Off	Off	Self-refresh
	S5	-	-	Off (0v)	-	Off	Off	Off	Off

Figure 2-13. *Server CPU power management example*

Summary

It is quite common for data centers to operate at less than full capacity for a large percentage of time. Power costs contribute a large percentage of the TCO of many data centers. The benefits of saving power in the CPU are compounded by reducing cooling costs as well (discussed in Chapter 4). The features described in this chapter can save significant power and cost over the life of a data center.

P-states (voltage/frequency scaling) provide a mechanism to "dim the lights" when full performance is not required. This will increase the time to complete a task, particularly in workloads that require significant compute. However, in many cases, the time that a given transaction takes to execute on a given node is small compared to network latencies, hard drive accesses, and other overheads. Increases in the compute time on a node for that transaction can be a small fraction of the overall response time. On the other hand, some jobs and tasks are very latency sensitive and these increased response times can be undesirable.

CPU thermal management protects the CPU from dangerous temperature levels with a combination of P-states and T-states. Platform thermal management will be discussed in more detail in Chapter 4.

Turbo provides a mechanism for processors to take advantage of full capabilities of the platform and data center design by increasing the frequency beyond the base frequency in order to achieve higher performance. Even some of the most latency-sensitive customers are beginning to use Turbo due to the large potential for increased performance.

C-states, clock gating, and power gating provide a mechanism to "turn off the lights" when cores or even entire packages are not needed. Although wakeups take some time (generally <50 μs), these delays are not observable in many usage models. A favorite customer question is, "I turned off power management and my performance went down. What happened?" C-states can also increase performance in many workloads by allowing other cores to turbo up to higher frequencies.

It is not uncommon for data center managers to disable all power management to avoid performance degradation in their fleets. Although not all power management techniques are right for all users, many can save significant money by finding the right features for their particular deployment. Chapter 7 will discuss how to monitor the behavior of a system, and Chapter 8 will provide guidance on how to tune and configure a system for different types of usage models.

www.dbooks.org

CHAPTER 3

Memory and I/O Power Management

CPUs cannot operate effectively without memory to store working data and I/O interfaces to bring in data from drives and the network. Although the CPU is a common focus for power management and power efficiency discussions, in many systems the memory subsystem can contribute a significant power footprint. I/O is also important but tends to contribute a much smaller piece of the pie. This chapter will provide an overview of server memory architecture and how the power and thermal management techniques work. It will also discuss how power is managed for the other I/Os that provide the CPU with the data required for operation.

System Memory

Memory power can contribute a very large +-percentage of the overall platform power in some system designs. Different usage models require wide ranges of memory capacity, causing the importance of memory power to vary from user to user. Different types of memory can also have a wide range of power consumption. This section will provide an overview of memory architecture and how it impacts power consumption in the data center.

Memory Architecture Basics

Before we discuss the power management capabilities of server systems, it is important to understand the basics of how memory works and how power is consumed. Let's start at the high level. Sticks of memory, or DIMMs, are plugged into slots on the platform. Each slot is connected to a *memory channel*. Multiple DIMMs can be connected to the same memory channel, but when this is done, those DIMMs share the same command/data connection, and therefore allow for increased capacity, but not bandwidth. The number of DIMMs per channel is abbreviated as DPC (e.g., 1 DPC = 1 DIMM per channel).

When the CPU issues a read to memory, it generally[1] fetches 64 Bytes (B)[2] of data from a single stick of memory. Each physical address (PA) in the system is mapped to a specific channel/DIMM that is connected into the CPU. The read/write is issued on the command bus, and data is returned (if a read) or sent along with the write command on the data bus. The relevant DIMM on the channel determines that the command is for it and processes the request.

Data on DDR3/DDR4 is handled in Burst-Length 8 (BL8). This means that a single access (read or write) uses eight slots on the memory bus (see Figure 3-1). The memory bus is 8 B wide, providing 64 B of data across these eight *bursts* and runs at the DDR frequency. So, a given channel can provide *DDR Frequency* (*GHz*) * 8 (*Bytes/clock*) of memory bandwidth (in GB/s). Each of these eight bursts will acquire some data from multiple *devices* on the DIMM (the exact number depends on the type of DIMM).

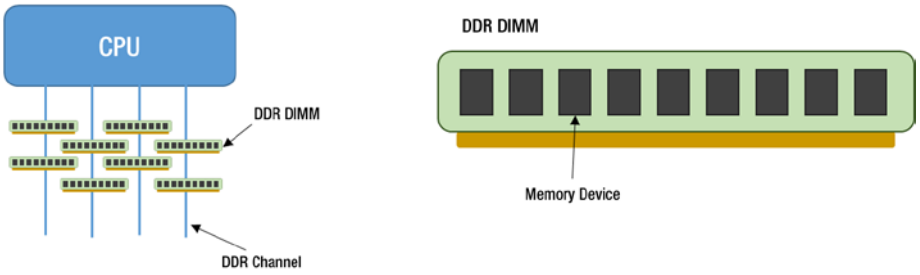


Figure 3-1. *DDR and the CPU platform*

Devices and Ranks

Figure 3-2 provides a high-level overview of memory DIMMs and how they connect to a CPU Socket. Each DIMM of memory consists of a number of memory devices. The devices are the actual "chips" that you will see soldered down to the DIMM. Each device supplies a subset of the 8 B chunks of data that are returned in each burst. Server memory devices/DIMMs can be ×4 or ×8 (called "by 4" or "by 8").[3] This refers to the amount of data that each device supplies toward each 8 B burst. ×4 memory supplies only 4 b of data for each 8 B chunk, and therefore 16 devices are required in order to supply the data. ×8 memory supplies 8 b of data, so only 8 devices are required. Device manufacturers commonly only produce a couple of device sizes at a time—and those devices can either be manufactured into ×8 or ×4 memory. ×4 devices allow for higher DIMM capacities using the same device size as well as improved reliability with error correcting code (ECC) by requiring more devices to supply data for a single 64 B access.

[1]Certain reliability features do exist, like memory Lockstep, which allow for a given 64 B chunk of data to be fetched from multiple devices in order to improve reliability. These are not commonly used in typical servers and are targeted at very high-availability systems.

[2]A bit (b) of data refers to a single binary piece of data (1 or 0). A byte (B) of data refers to a collection of 8 bits.

[3]Other types exist, but are not common in server usage models (e.g., client devices using DDR3 commonly supported ×16 memory as well).

www.dbooks.org

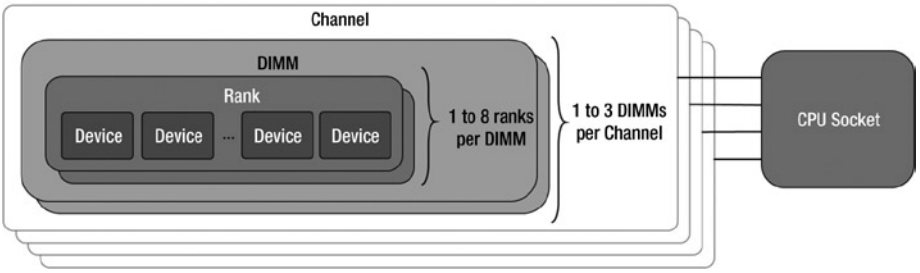


Figure 3-2. *DDR channels, DIMMs, and ranks*

DIMMs also have *ranks* (usually one, two, four, or eight). Individual DRAM devices are single-ranked. So, when a DIMM supports two ranks, for example, this means that the number of DRAM devices on that DIMM is doubled. So, a ×8 DIMM that requires 8 devices to supply 64 B of data will actually have 16 devices if it has two ranks, or 32 devices if it has four ranks. A single bus connects the DIMM to the CPU, and all the ranks on that DIMM share that bus. However, each rank is able to operate somewhat autonomously from the others. One of the biggest challenges for a DRAM device is switching between doing reads and writes (and back again). As a result, the memory controller must insert a sizeable bubble between these types of transactions to a given rank. DIMMs that support multiple ranks are able to sneak other transaction onto the channel bus while one rank switches modes, allowing for improved bandwidth and average latency. As a result, for performance, we generally recommend that you use either two single-ranked (SR) DIMMs of the same size on a channel, or one or more dual-ranked (DR) DIMMs in order to help hide these inefficiencies. Quad-ranked (QR) and even some oct-rank (OR) DIMMs also exist on the market.

■ **Note** All memory is not equal. As with CPUs, new process technologies are being applied to memory, which enables lower power and increased capacities. At a given point in time, vendors are only able to economically manufacture devices up to a given size. The "next size up" (two times the capacity) are at times available, but they come at a significant cost premium. By increasing the number of ranks or moving from ×8 to ×4 devices, vendors can increase the number of devices on a DIMM and thus increase capacity. Using a smaller number of higher capacity devices can consume significantly less power than using a large number of lower capacity devices, despite the fact that both can provide the same memory capacity.

Memory Error Correction (ECC)

Server memory typically leverages ECC memory to provide protection, both from transient errors and device failures. Each 8 B burst of data that is supplied includes an additional ninth byte of data that provides the ECC protection. This increases the cost and power of memory proportionally but is generally considered a "must-have" in server deployments.

■ **Note** It is not possible to disable the ECC device in order to save power in systems today.

With ×8 memory, an additional device is included on the DIMM to support ECC. With ×4, two additional devices are required. ×4 memory provides improved reliability because each device provides a smaller percentage of the data. If a single device fails, the ECC algorithm is able to correct all data and continue to operate the system. On ×8 memory, if a device fails, it is possible to detect such a condition and hang the system, but correction is not possible.

Memory Capacity

The capacity of a DIMM is a function of the ranks, devices, and device size. DIMMs are typically sold in Gigabytes (GB), whereas devices are typically referred to in Gigabits (Gb). The following formula summarizes how one can calculate the capacity of a DIMM based on the components:

$$Capacity\,(GB) = \frac{\left(DeviceSize\,(Gb) * Ranks * \frac{Devices}{Rank}\right)}{\left(\frac{8\ bits}{Byte}\right)}$$

$$= \frac{\left(DeviceSize\,(Gb) * Ranks * \frac{64B}{byX}\right)}{\left(\frac{8\ bits}{Byte}\right)}$$

$$= \frac{\left(DeviceSize\,(Gb) * Ranks * 8\right)}{byX}$$

Examples:

- DR ×8 4 Gb = 4 Gb * 2 ranks * 8 / (×8) = 8 GB
- QR ×4 8 Gb = 8 Gb * 4 ranks * 8 / (×4) = 64 GB

One can increase DIMM capacity by increasing the number of ranks, the device size, or the number of devices per DIMM (moving from ×8 to ×4).

www.dbooks.org

■ **Note** For optimal performance, we typically recommend that you use dual-ranked (DR) memory, because it allows for more efficient use of the memory bus. This is particularly true of workloads that make use of high memory bandwidth. Single-ranked memory can also show good performance when you use it with multiple DIMMs of the same size per channel (since the memory controller has more than one rank to work with). Quad-ranked (QR) memory also can make efficient use of the memory bus, but it typically requires a lower frequency.

Device Power Characteristics

Many users have a feel for how much memory they think they need, but it can be difficult to understand how to populate the system in order to provide that desired memory. Table 3-1 provides some general rules of thumb for how memory power scales. In each of these cases, capacity is increased by either 1.5 or 2 times. You might expect power to scale directly with capacity, but different decisions result in different power impacts. (Note that these numbers are intended only as a conceptual guidance, not as a hard rule.)

Table 3-1. *DDR4 DIMM Power Scaling Examples*

Parameter	Power Impact	Capacity	Other Notes
Single-rank to dual-rank	~1.3–1.5 times	2 times	Improved performance with a one-DPC configuration, particularly with high-bandwidth workloads.
×8 to ×4	~1.4 times	2 times	Improved reliability.
Two times device capacity	<1.1 times	2 times	Device size increases commonly come with better process technology, so this is difficult to accurately quantify.
One DPC to two DPC	~1.5–1.7 times	2 times	Improved performance with single-ranked DIMMs. Can decrease memory frequency. Power does not double in this case, because the bandwidth provided by each DIMM is 50% of the bandwidth if two DIMMs were used.
Two DPC to three DPC	See note	1.5 times	Generally has significant impact on memory frequency. Not a fair comparison. Three DPC should be used for customers who need capacities not economically possible with two DPC.

■ **Note** Using dual-ranked memory in a one-DPC configuration is typically the most power/performance efficient across a range of workloads. Increasing the number of DIMMs per channel tends to be less power efficient than other alternatives but can also be attractive from a DIMM cost perspective. If low capacity and low bandwidth are required, one-DPC single-ranked topologies are the most power efficient. However, this efficiency quickly falls off if memory bandwidth begins to get stressed. Before purchasing one-DPC single-ranked systems, we highly recommend that you characterize the bandwidth requirements of their workloads first (as described in Chapter 7).

DDR has been optimized to minimize leakage power. Not only does this result in minimal power scaling with temperature, but it also minimizes the power cost of increasing the device capacity. This tends to be the most power-efficient mechanism for increasing capacity but can also be price prohibitive, especially after a certain point.

Power deltas for additional ranks and DIMMs tend to be smaller at higher bandwidths since the overheads are amortized across the power for providing the necessary bandwidth. This is not the case with ×8 to ×4 scaling.

DDR3 vs. DDR4

At a high-level, the architecture of DDR3 and DDR4 are very similar. From an end-user perspective, DDR4 enables higher frequencies while running at lower voltages and consuming less power. There are some other internal changes for improving performance (e.g., more banks). Table 3-2 shows how memory technology has progressed in recent years. Voltage has decreased despite increases in maximum frequency. Larger and larger devices have also been possible as process technology shrinks. Note that the maximum device capacities do not always represent what a given processor can support.

Table 3-2. *DDR Generation Comparisons*

DDR Generation	Voltage	Frequencies	Device Capacities
DDR2	1.8 V	up to ~1600	up to 1 Gb
DDR3	1.5 V	up to ~1866/2133	Spec supports up to 8 Gb (4 Gb common)
DDR3L	1.35 V	up to ~1333/1600	
DDR4	1.2 V	up to ~3200 (TBD)	Spec supports up to 16 Gb

DDR3 supported two different voltages: 1.35 V and 1.5 V. However, running at the lower frequency reduced the peak frequency that could be achieved. DDR4 transitioned all memory to use 1.2 V but also provided a significant boost in peak frequencies (and with it, peak bandwidth).

www.dbooks.org

DDR voltage plays a significant role in energy efficiency (the exact amount varies a good amount across memory types). On processors supporting DDR3/DDR3L memory, a tradeoff could generally be made between selecting DDR3 memory and achieving a higher frequency, or DDR3L and operating at a lower voltage. Higher frequencies can significantly improve memory bandwidth and performance on certain workloads (particularly in the high performance computing space). They also provide slightly lower latencies, but these benefits tend to be small (a few percent of performance at best). Many enterprise systems with large memory capacities come nowhere close to the memory bandwidth limits of the system and can save significant energy by using DDR3L with minimal impact to performance. In platforms with smaller memory topologies, the importance of memory voltage overall is much smaller.

RDIMMs, UDIMMs, SODIMMs, and LRDIMMs

Most servers typically use *registered DIMMs* (RDIMMs), although *unregistered DIMMS* (UDIMMs) are an alternative. Signal integrity is challenging with high-speed memory, particularly when multiple memory DIMMs coexist on the same memory channel. RDIMMs include a register on the DIMM that reduces the electrical load on the memory controller; this improves signal integrity and allows for increased memory frequencies (and higher performance). RDIMMs have traditionally been more expensive than UDIMMs because of the smaller volume and additional components. However, this trend may or may not continue as more client devices move to different memory technologies than those found on servers. The register also consumes measurable power (on the order of ~0.5 to ~1 W per DIMM) when active (some of this power can be saved in low-power states). Different processors and platforms have varied rules and constraints about the maximum frequency that can be supported by different types of memory (UDIMM vs. RDIMM) as well as the topology of memory (number of ranks, number of DIMMs per channel, etc.).

UDIMMs can be purchased with and without ECC. ECC increases the number of devices required by 12.5%, and power increases at about the same rate. Small-outline DIMMs (SODIMMs) show very similar characteristics to UDIMMs (both ECC and non-ECC)—they are just physically smaller and therefore cannot hold as many devices.

ECC UDIMMs and SODIMMs are a good solution for low capacity, low power, and low cost deployments. Systems requiring larger capacities or high reliability typically use RDIMMs.

LRDIMMs (load-reduced DIMMs), a type of RDIMM, provide some additional buffering that allows them to provide access to a large number of devices and still maintain high frequencies. QR RDIMMs suffer from electrical issues that limit their frequencies. LRDIMMs attempt to address this by providing an additional buffer chip on the DIMM to address the increased device count and improve signal integrity. LRDIMMs provide high capacity and high performance. The additional buffer chip consumes some additional power and increases memory latencies slightly, but neither is really of consequence at the platform level. LRDIMMs are generally more expensive per GB of memory compared to normal DR RDIMMs. When they were originally released, they also had a healthy price premium over high capacity QR RDIMMs, but that price delta has come down over time (on DDR3). If you need massive memory capacity, LRDIMM is a good place to start.

Memory Channel Interleave and Imbalanced Memory Configurations

Each channel has a finite amount of memory bandwidth that it can sustain. By interleaving a stream of requests across multiple channels, high memory bandwidth (and higher performance) can be achieved. In order to get optimal performance in a system, each channel should be populated with the same capacity of memory. However, sometimes this is not the most cost effective way to achieve a given desired memory capacity. If imbalanced configurations are going to be used, it is best to avoid situations where a single channel has more capacity than the others, because this results in a section of memory with a "one-way" interleave (and 25% of the theoretical peak bandwidth) as shown in Figure 3-3.

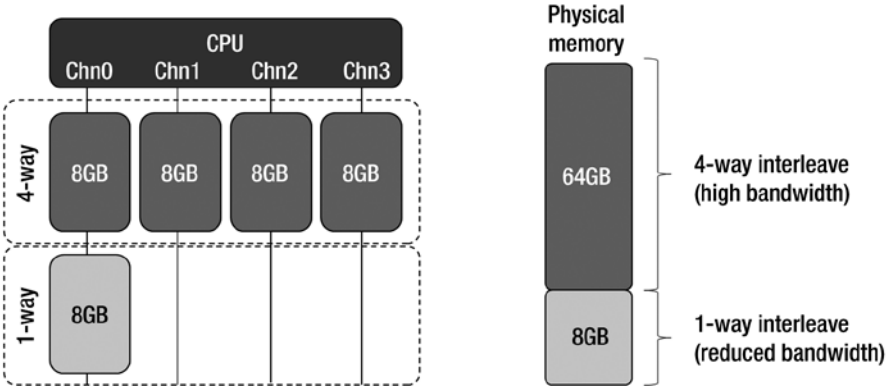


Figure 3-3. *Imbalanced memory interleave example*

By default, most systems today are set up to have separate non-uniform memory access (NUMA) memory regions assigned to each socket. For example, a system with 32 GB of memory would have the first 16 GB of memory allocated on socket 0 and the second 16 GB on socket 1. An alternative to this approach is to use a uniform memory access (UMA) allocation, which interleaves every other cache line across the sockets (effectively providing a single 32 GB region across both sockets in the previous example). In many usage models, this is detrimental to performance because it increases the latency of half of the requests by forcing them to the remote socket. Many users are better off letting the OS (which is aware of this behavior) manage memory allocation and attempt to locate memory on the local socket in order to reduce memory latency. However, certain usage models do exist where the memory NUMA schemes are unable to successfully locate memory in the optimal place and actually lose performance by trying to do so. This is, however, generally uncommon, and most workloads either benefit from such an allocation or show little sensitivity.

Memory interleave does not typically have a direct impact on memory power savings. Power efficiency here is typically achieved by optimizing for performance. Some interesting effects are possible, although in imbalanced configurations. For example, imbalanced configurations can result in certain ranks being accessed infrequently, resulting in higher CKE power savings (discussed more in the following pages).

www.dbooks.org

Power and Performance States

A number of power savings techniques[4] exist for reducing memory power when it is not fully utilized. In general, most of these techniques fit into the category of "turning things off" and not "turning them down." There are really two main techniques for saving power:

- Turning off CKE (clock enable): Power savings during short idle periods at the rank granularity
- Self-refresh: Power savings during long idle periods at the channel granularity

Self-refresh allows for significant memory power to be saved but also can require non-trivial wakeup costs (~10 μs). Turning off CKE provides less power savings but can have very fast wakeups (~10 ns). Turning off CKE can also be done on a rank-by-rank basis, whereas self-refresh must be performed at the channel granularity. As a result, in servers, self-refresh is typically targeted at idle systems, whereas CKE is targeted at moderately active systems. Dynamically managing memory frequency at runtime has not been productized. Changing frequency is a non-trivial piece of work, and the power savings are generally not significant due to the static voltage.

CKE Power Savings

Each rank has a clock enable (CKE) signal that is driven from the CPU memory controller to the DIMM. By de-asserting CKE, the rank is allowed to enter a low power state that can be exited quickly (~10 ns to ~100 ns). A number of different flavors of CKE have differences in their details, but in general, they mostly behave the same. Because CKE is managed on a per-rank granularity, there is potentially more opportunity for CKE power savings on systems with more ranks.

At a high-level, there are two types of CKE:

- Active power down (APD): Memory pages are kept open[5] and the row buffer stays powered up.
- Precharge power down (PPD): The memory pages in all banks on a rank have been "closed" or "precharged," and the row buffer can be powered down.

At first glance, this may sound like a simple power/performance tradeoff. APD saves a bit less power but keeps pages open. However, in practice on servers, it does not actually work out this way. Many times when a rank goes idle for long enough to turn off CKE, the memory pages are also finished being accessed, and therefore having them

[4]See the DDR3 and DDR4 specifications at www.jedec.org for more details. JESD79-4A contains information for DDR4, and JESD79-3F for DDR3.

[5]Memory pages are different from software/TLB pages. Different device types have different memory page sizes. A single 4 K software page can be mapped to either a single memory page (in open page configurations) or to many memory pages (in closed page topologies). Large pages (2 M and larger) typically exist over multiple memory pages.

closed is good for both performance and power. Both CKE PPD and CKE APD are able to save on the order of 30% of the power, and the differences between the two for both power and performance are negligible.

The CPU can force PPD to be used by issuing a PREALL command to a given rank before de-asserting CKE. This closes all the pages in all banks on the rank, allowing PPD to be used. Alternatively, the CPU can simply de-assert CKE when all necessary timing parameters have been met. If one or more pages are open, the DIMM will be in APD. Otherwise, it will be in PPD. In Intel server documentation, the *PPD mode* refers to the case where PREALL is explicitly issued before de-asserting CKE, while *APD mode* disables this PREALL. It is still possible to get into a PPD state from the APD mode if all pages happened to be closed at the time CKE was de-asserted.

On DDR3, there were two main versions of PPD: PPDF (fast) and PPDS (slow). PPDS saved more power at the cost of a slight increase in exit latency. Usually the added exit latency is trivial, so PPDS is generally the better state. In the big picture, the differences between PPDF, PPDS, and APD are not large (either for performance or power efficiency).

One of the big changes with the transition from DDR3 and DDR4 is with how ODT is handled. Rather than requiring the memory controller to manage ODT, it is handled autonomously by the DIMM. In addition to this, the PPDS and PPDF states have been merged into a single PPD state where the DLL is kept powered and ODT is managed by the DIMM. This new simplified mode has excellent power savings. The DLL was redesigned on DDR4 and consumes much less power when active. It is possible to turn off all the DLLs on a channel when an entire channel is idle and save additional power. However, this state has not been productized on Intel servers thus far.

When all ranks on a DIMM are powered down, the register on RDIMMs can also enter a low power state. This does not save all of the register power but can save a couple hundred mW per DIMM.

It is also possible to power down the IBT (input-buffer termination). IBT OFF theoretically can save ~100 mW per DIMM with CKE (it also exists with self-refresh). However, in practice, the savings tend to be much smaller at the platform level because of the increased exit latency. This mode has not been aggressively enabled on servers due to the low power savings upside and wake latency exposures. It is more interesting in microserver usage models, particularly with self-refresh (more later).

Table 3-3 provides a summary of the various types of CKE power savings, including on which types of DDR they are supported. Table 3-4 provides a summary of how CKE has evolved over multiple processor generations.

Table 3-3. *CKE Mode Summary*

Type	Granularity	Banks	ODT	DLL	DDR3	DDR4
APD	Per rank	≥ 1 active	On	On	Supported	Supported
PPDF	Per rank	All precharged	On	On	Supported	Supported (PPD)
PPDS	Per rank	All precharged	Off	Off	Supported	Not Supported

www.dbooks.org

Table 3-4. *CKE Across Generations*

Generation	DDR	APD	PPDF	PPDS	Channel PPDS-DLLOFF
Nehalem/Westmere E5	DDR3	Y	Y	1 DPC	N
Sandy Bridge/Ivy Bridge E5	DDR3	Y	Y	Y	Y
Avoton	DDR3	Y	Y	1 DPC	N
Haswell E5	DDR4	Y	N	N	N

Self-Refresh

DRAM devices (unlike SRAM) must be periodically *refreshed* in order to keep the data valid. Refreshing memory is really nothing more than reading it out of the arrays and writing it back in. During normal operation, the memory controller is periodically issuing refresh commands in order to refresh a portion of the device. The entire device must be refreshed periodically (usually on the order of 10s of milliseconds). When a given channel is not being used, it is possible to put all the DIMMs on that channel into a *self-refresh* state where the DIMM itself is responsible for handling refresh. This state both saves power on the DIMM and allows for additional power to be saved in the CPU memory controller and I/Os. However, this additional power savings generally comes with a non-trivial latency cost. Like with CKE, there are different flavors of self-refresh that provide varied power savings and latency characteristics.

Because self-refresh is performed at the channel granularity and because it tends to have longer exit latencies, it is typically used for saving power when the system is completely idle. Self-refresh residencies in active systems tend to be very low. Typical high-capacity server DIMMs that are in self-refresh tend to consume on the order of 0.2 W to 0.5 W.

The main differentiator between the different flavors of self-refresh is how the CK signals are handled. This is referred to as the *clock stop mode*. CK and CK# are a pair of differential clocks that are necessary for transmitting commands and data between the CPU and memory. If the CPU continues to drive these signals during self-refresh, the wakeup latency can be relatively fast (< 1 μs). However, this mode saves minimal additional power compared to simply turning off CKE and the DLL. The clock can also be stopped. It can be tri-stated, driven low, or driven high. Each of these states results in additional power savings, but exit latency increases to ~10 μs.

CKE can also be tri-stated (i.e., not driven to either 0 or 1) during self-refresh to save some additional power (compared to driving it low). It must be driven low on UDIMMs, but otherwise it can be tri-stated (the voltage is not driven high or low; it is simply left to float to wherever it settles).

■ **Note** Self-refresh is most useful when the entire system is idle and CK can be stopped. As a result, it is used sparingly when cores/IO are active (where CKE management is used instead). However, when the entire system is idle, it can be used more aggressively. This is particularly the case when it is paired with package C-state power savings features. In these cases, the 10 µs wake latency can frequently be done in parallel with other long-latency operations (ramping voltage, locking PLLs, etc.), making the power savings effectively "free."

Voltage/Frequency

Systems today will not dynamically change the voltage/frequency of DDR. On DDR3, some devices support running at both 1.5 V and 1.35 V (called DDR3L). With the transition to DDR4, all DIMMs run at 1.2 V. DDR3L was released after DDR3, and DDR4L is expected in the future as well.

Running DDR3 at 1.35 V generally exhibits significant memory power savings. The amount/percentage is very sensitive to the configuration in question, but using DDR3L can save significant power on systems that leverage a large amount of memory.

On the other hand, generally the frequency of memory is really not all that important. Running at lower voltages can limit the achievable frequency in the system on DDR3, and the frequency can impact the maximum amount of power that a DIMM can consume, but the power to run most workloads is typically not that sensitive to frequency. As an example, taking some DDR3L memory that typically runs at 1333 and decreasing the frequency to 1066 and running at the same (moderate) throughput would save less than 5% memory power. At the same time, such a change could also reduce memory CKE residency or increase core active time, further reducing the power benefits from the reduced frequency. With that said, running at 1333 does provide an additional 25% bandwidth, and if that bandwidth is actually used, then the memory power will increase by ~10%–20%. However, this is generally a great power/performance tradeoff—25% more used bandwidth usually means 25% more performance. The 10%–20% memory power increase for 25% more performance is a small power price to pay when measured at the wall.

On DDR4, the percent power savings by reducing frequency is larger, but this is largely because the overall power has gone down. Power savings with DDR4 is typically on the order of 50–400 mW per DIMM when reducing by a single frequency bin (again, without taking into account additional power consumed elsewhere as a result of the lower performance). Long story short: reducing memory frequency is generally not a good idea.

DDR Thermal Management

Managing the temperature of memory DIMMs is critical to preventing loss of data or system crashes. Most server memory is capable of monitoring temperature, but the CPU is responsible for providing the thermal management algorithms that protect the DIMMs. Memory temperature is another input to the fan speed control algorithms that are discussed in Chapter 4.

www.dbooks.org

Monitoring Temperature

The DIMMs themselves typically provide a thermal sensor called a thermal sensor on-die (TSOD), which provides a single temperature reading for an entire stick of memory. Historically, not all memory used in servers included a TSOD (UDIMMs in particular), but as time has progressed it has become standard. There is only a single thermal sensor on a DIMM, and it is commonly located near the center of the DIMM (lengthwise). Air commonly flows down the DIMM and heats up as it passes over the devices. As a result, the first device tends to be at a lower temperature than the last device, with a single temperature reading taken somewhere in the middle. The CPU (or BMC) reads this temperature over a System Management Bus (SMBus).

Memory Throttling

The CPU is responsible for throttling requests to the DIMM in order to reduce the memory temperature when it begins to enter a high temperature range. Table 3-5 provides a summary of some of the common methodologies for management memory thermals.

Table 3-5. *Memory Thermal Management Techniques*

Mechanism	Requires TSOD	Description
OLTT	No	**Open-Loop Thermal Throttling** OLTT is the simplest mechanism for managing thermals. A static bandwidth limit is put in place in an attempt to avoid high-power operation.
CLTT	Yes	**Closed-Loop Thermal Throttling** CLTT takes temperature readings from the TSOD and performs varied levels of memory bandwidth throttling in order to keep the DIMM in a safe operating range.
Dynamic CLTT	Yes	**Dynamic CLTT** Dynamic CLTT is an enhanced version of CLTT that takes other platform information into account (such as fan speed) to adjust the throttling configuration dynamically to save additional power.

OLTT (open-loop thermal throttling) is the most basic mechanism for performing throttling. Historically it was used in low-cost systems that did not have TSODs available on the DIMMs to provide temperature-based throttling. TSODs are standard on most server memory today, but the legacy OLTT mechanisms are still available.

CLTT (closed-loop thermal throttling) is the standard mechanism for providing memory temperature protection. The CPU monitors the temperature of the DIMMs and engages varied levels of throttling depending on the temperature. Doubling the memory refresh rate is also commonly performed at higher temperatures in order to avoid data corruption.

As discussed previously, temperature increases as air flows down the length of a DIMM. This increase is called a *thermal gradient*. The amount of gradient can vary with other platform parameters. For example, high fan speed results in more air flow and smaller temperature gradients than reduced fan speeds. With the baseline CLTT support, platform designers must assume some amount of gradient when configuring the CLTT throttling algorithms. The CPU provides an interface with Dynamic CLTT for the platform management firmware to dynamically change the throttling constraints based on an estimate of the thermal gradient. This can be used to save power or to prevent throttling when fan speeds are high and there is a smaller gradient. The algorithms used to estimate the gradient and configure the throttler are typically proprietary IP for a platform designer.

MEMHOT is a platform signal similar to PROCHOT. Different products make different use of MEMHOT. It can be used as an input to the CPU, providing an indication from the platform that the CPU should perform memory throttling. This input can be used for the platform to trigger memory throttling when a thermal issue is detected in the platform (even if the DIMMs themselves are not too hot). It is also frequently used to throttle memory power/thermals when some other undesirable event is detected in the platform like an overheating power supply. MEMHOT can be used as an output from the CPU and as an indication to the platform management that the DIMMs have reached a high temperature. On some CPUs, MEMHOT can also be bidirectional and support both input and output modes simultaneously.

DDR3 and DDR4 memory also supports an EVENT# pin that triggers when the TSOD detects high temperatures. This open-drain pin is typically wired directly to the BMC and is not used directly by the CPU. It is commonly used for detecting critical temperature levels that require an immediate system shutdown.

CPU DDRIO

I/Os exist on the CPU that connect to the traces that go to the DIMMs. These I/Os typically run at the same voltage as the memory and are supplied by the same voltage regulator.[6] Multiple channels frequently share a voltage regulator. DDRIO power at a first order is a function of the bandwidth that it is driving (both reads and writes). There is some additional power cost that results from increasing the number of DIMMs, but this is not a first-order impact. Despite the fact that DDRIO power shares a voltage regulator with memory, the power is typically assigned to the CPU for Running Average Power Limit (RAPL) usage models. This is done in order to effectively manage thermals within a power budget. It also allows the CPU to trade off unused power (and thermal) headroom back to the CPU cores when underutilized. This can be useful, since many high DDRIO power workloads do not require heavy core power, whereas core-centric workloads tend to have low to moderate DDRIO usage.

[6]Platforms that leverage buffered memory solutions (such as Haswell EX) have more complicated power delivery designs, and may run the DDRIO and DIMMs on separate voltage regulators.

www.dbooks.org

Workload Behavior

Workloads tend to either demand very high memory bandwidth (≥ 80% peak) or be relatively insensitive to memory throughput (< 30% peak). There are always exceptions to the rule, but this is a trend that can be observed across a range of server workloads. Many of the workloads that fit into the high bandwidth category come from the high-performance computing segment, or could benefit from data structure optimization to improve cache locality. Memory power has a moderate dynamic range even without memory power management features. This is particularly the case with one-DPC configurations. It tends to be less apparent with two-DPC and three-DPC configurations, since on average, the percentage of traffic that goes to a given DIMM is cut (in half or in a third), reducing the actual dynamic range of the DIMM bandwidth. As an example, with a one-DPC DR configuration, scaling bandwidth from 20%–80% increases memory power by ~1.5 times.

Memory Reliability Features

A number of reliability features exist for memory that can have an impact on the power drawn by a given workload.

Memory Lockstep is a reliability feature where a single 64 B piece of data is stored across two DIMMs on two different memory channels. Since DDR3 and DDR4 work in BL8 mode, a single read or write actually fetches 128 B of data from the memory, increasing the amount of memory bandwidth that most workloads will consume. Lockstep tends to only be used in environments where high reliability is required because it both increases memory power and tends to have a measurable performance impact.

Patrol Scrub is a memory reliability feature that is typically enabled on all server CPUs by default. This feature attempts to walk through all of the memory space more or less every 24 hours, reading each line and checking the ECC. The goal is to identify errors while they can still be corrected. A single channel on each socket is typically scrubbed at a time. In certain situations this can result in channels not entering self-refresh because this blocks scrubbing, thus increasing memory power of idle systems. Patrol scrub is generally a low-cost method for reducing exposure to uncorrectable errors, and the added power cost is generally worth that reduction in exposure.

CPU I/Os

In addition to memory, there are a number of additional I/O capabilities that exist on modern server processors including interconnects that connect multiple sockets together (such as Intel QPI) as well as PCIe, which provides connectivity to devices like network cards and storage.

CPU Interconnect

In multi-socket systems interconnects exist that connect the different sockets to each other. These interconnects are used to maintain coherency across the sockets, to provide a communication channel between the sockets, and to connect memory that is connected from one socket to the other. In order to provide high performance and

prevent the coherency overhead from slowing down the performance of the cores, these interconnects are required to be high bandwidth and low latency and consume a non-trivial amount of power.

On the Sandy Bridge EP processors, the QPI interconnect consumes ~5 W of power per socket. Much of this power is consumed in the I/Os, and therefore it does not scale down significantly from one process generation to the next.

Because of their moderate power draw, these interconnects are most efficient on higher power processors that can amortize the cost of the power. Using a 5 W multi-socket coherent interconnect to hook up two 20 W processors is typically not worth the overhead. Rather than spending power on the I/Os, you are better off simply using a higher power single socket processor.

Link Power States

Power management of an interconnect is no different from anything else at a high level. However, one typical constraint is that it is difficult to scale the voltage of an interconnect in order to efficiently scale the frequency. As a result, reducing frequency can save power, but this is not always the most efficient decision. Table 3-6 illustrates some of the power states available on Intel's QPI 1.0.

Table 3-6. *QPI 1.0 Link Power States*

State	Name	Power	Granularity	Description
L0	Link Active	100%	–	Link active and running at full size and frequency. Provides maximum bandwidth at minimum latency.
L0s	Link Sleeping	~50%	Per direction	Subset of lanes asleep and not actively transmitting data. Not possible to send any information. Some lanes (clocks, etc.) still active, allowing for fast wakeup.
L0p	Partial Link Active	~75%	Per direction	Similar to L0s state, but a subset of the data lanes remain awake (typically half, but anything is possible). Bandwidth is reduced and latency for transmitting data increases.
L1	Link Down	< 10%	Entire link (both directions)	Link is completely powered down. In order to transmit data, it must be retrained.

www.dbooks.org

L0p is a very useful power management state, particularly at low system utilizations. Many server workloads, particularly in the enterprise area, do not make heavy use of either memory bandwidth or the interconnect bandwidth. As a result, the loss in bandwidth from cutting the link in half has minimal impact on the actual performance or throughput of the system. L0p does, however, increase the amount of latency that it takes to transmit a data packet from one socket to the other. Data packets typically contain 64 B of data, and the link itself is much smaller than this. This can add ~10 ns of latency for such transfers. Although this latency is mostly inconsequential at low system utilizations, it can cost 1%–2% peak performance on workloads that are very latency sensitive and transmit significant data from one socket to the other.

L1 is an excellent state for idle systems. Although it takes multiple microseconds to wake a link back up, this is commonly "free" if there are long-latency actions being performed in the system (memory self-refresh, ramping voltage from a retention level to an active level, etc.). As a result, L1 is typically used during package C-states. Using it more aggressively during active states tends to result in performance glass jaws and even platform power increases.

L0s was a state that was productized on early QPI generations, but it has since been not supported. L0s is theoretically most useful if workloads exhibit bursty behavior between being active and completely idle. With a coherent interconnect and server workloads, it is uncommon to find periods of *no* traffic that last longer than a few hundred nanoseconds unless the system is completely idle. A trickle of coherency and communication traffic between sockets always seems to exist, particularly in real workloads that do not exhibit perfect NUMA locality. In situations where the system is indeed idle, L1 provides the necessary idle power savings.

■ **Note** L0s support has been removed from recent CPUs due to minimal power savings upside. L1 is only used during package C-states, where its latency can be hidden by other components during a wakeup.

Dynamic control of QPI frequency is not performed today. By reducing the frequency of the interconnect, not only is the bandwidth reduced, but the latency for transmitting data packets increases. This is particularly the case with L0p. This impacts the performance of the cores, which can spend more time stalled waiting for data to be returned to them. Not only does this impact the peak performance of the system, but it can even reduce the power/performance efficiency across a range of utilizations.

PCIe

The PCIe specifications provide standardized mechanisms for saving power. These link states are used across the wide range of devices that make use of PCIe (from low-power devices to servers). When it comes to PCIe link power management, server CPUs today are typically slaves to the devices that are connected to them. The devices themselves (through their own dedicated driver/firmware/hardware) initiate the transitions into power management states. The CPU is able to send negative-acknowledgment (NACK) requests but never initiates them.

Link Power States

PCIe uses L0s and L1 states. Unlike QPI, there is no defined partial width (L0p) state. However, it is possible to dynamically change (or statically configure) the link width that a device uses through the *upconfigure* flow. Table 3-7 provides an overview of the power management states used by PCIe.

Table 3-7. *PCIe Link Power States*

State	Name	Savings	Exit Latency	Granularity	Description
L0	Link Active	–	–	–	Link is active and running at full size and frequency. Provides maximum bandwidth at minimum latency.
L0s	Link Sleeping	~20 mW per lane, per direction	Microseconds	Per direction	Subset of lanes asleep and not actively transmitting data. Not possible to send any information. Some lanes (clocks, for example) are still active, allowing for fast wakeup. L0s is initiated autonomously by the link layer (no OS/driver interactions).
L1	Link Down	~100 mW per lane	Microseconds	Entire link (both directions)	Link is powered down. In order to transmit data, it must be retrained. Can be used dynamically at runtime. L1 can be triggered both autonomously by the connected device (ASPM L1) or through an OS/Driver call (L1-soft).
L2	Link Off	~125 mW per lane	Milliseconds	Entire link (both directions)	Saves slightly more power than L1. Generally used for unconnected links and links that are disabled at boot. L2 is only initiated through a software request.

www.dbooks.org

■ **Note** Power savings is design dependent. These numbers provide a reference point.

L1 can be used during idle periods but has typically not been heavily utilized because of the latency impact and minimal power savings at the wall relative to overall socket power. Drivers and devices are typically tuned to use this state in only the most-idle conditions to avoid wakeup cost. With deep package C-states, the wakeup cost of L1 can generally be hidden since the CPU is informed about the wake and can perform other wakeup actions in parallel. L1 is becoming more interesting in the microserver space where the I/Os contribute a much larger chunk of the node power.

■ **Note** L0s is not supported on recent server processors. It saves relatively small amounts of power with non-trivial exit latencies. It is generally better off leaving a port in L0 or allowing it to drop all the way to L1.

Link Frequency/Voltage

PCIe has gone through three generations. Each generation has a single specified frequency at which the device runs (see Table 3-8). Multiple voltage/frequency points are not supported. However, the newer generation devices support (by rule) backward compatibility to the prior generation's frequencies. Changing frequency requires a full link retrain and is typically not performed dynamically at runtime today (although it is supported). Voltage is constant across generations/frequencies. When a PCIe 2.0 device is connected into a processor that supports PCIe 3.0, the device can only operate in PCIe 1.0 or 2.0 modes.

Table 3-8. *PCIe Generations*

Generation	Frequency	Theoretical ×8 Bandwidth
PCIe 1.0	2.5 GHz	2 GB/s
PCIe 2.0	5 GHz	4 GB/s
PCIe 3.0	8 GHz	7.88 GB/s

Although PCIe 3.0 only increased frequency by 60%, it was able to almost double the peak throughput. This was due to more efficient encoding and better use of the available wires for actual data.

Link Width

The number of lanes of PCIe impact the bandwidth that can be pushed through a link. Double the lanes, and the theoretical peak bandwidth is also doubled. The maximum number of lanes that a device is able to use is a function of both the device itself as well as the slot that it is connected to. Some systems may also allow the width of certain slots to be configured by BIOS. This can save a small amount of power but is generally not significant and can significantly reduce throughput. This could be useful for benchmarking but is not something that would generally be recommended for production systems.

PCIe devices can dynamically change the number of lanes in use at a given point in time. This can be done through software drivers (through the *upconfigure* and *downconfigure* flows) or through autonomous linkwidth change, allowing them to save power when lower bandwidth is required. These flows can be thought of as a way to reconfigure the link at runtime by restarting it with a different width (the lanes that are no longer used are no longer driven, reducing power). Reconfiguration typically takes microseconds, during which time the link is unavailable to transmit data. This flow has not been aggressively productized in server PCIe devices to date. These modes typically save relatively small amounts of overall system power and can cause non-trivial impacts to performance and latency in server usage models.

Hot Add

Many servers support Hot-Add flows. These allow PCIe cards to be inserted into the system at runtime without a reboot. However, this comes at a cost. In order to support Hot-Add, the lanes periodically cycle through a *DETECT* state that consumes moderate power. For a ×8 lane, this can add on the order of 100 mW of power on average. Through BIOS, PCIe lanes can be forced into an L2 state so that they do not perform this detection.

D-states

Similar to core C-states, PCIe devices can use D-states that indicate that a device is powered down. D-states are commonly used in phones, tablets, and laptops under idle conditions. They are not common under active load in servers. D-states are traditionally handled outside the CPU by PCIe devices that are connected to the platform with no interaction with the CPU (except side effects like the L1 state being used). As traditionally discrete devices are integrated into SoCs, this may change.

Summary

Memory can consume a large percentage of the overall power “pie” in many server systems. This is particularly the case in deployments that depend on large memory capacities. CKE and self-refresh can save significant amounts of power with almost no impact to the performance of the system. Making use of these capabilities is critical for achieving power efficiency in deployments with large memory capacities.

www.dbooks.org

Selecting the correct type and configuration of memory can also have a large impact on energy consumption as well as performance. Building systems with at least two ranks per channel tends to result in the best performance across a wide range of workloads. Larger capacity devices tend to provide additional capacity at lower power cost than more DIMMs or more ranks, but they can also be cost prohibitive.

I/O power has historically been a much smaller contributor to overall system power, and getting overly aggressive with power optimizations in this area can be counterproductive. This is particularly the case with high-power CPUs. I/O power does become much more significant on CPUs with low power draw such as microservers and embedded devices.

CHAPTER 4

Platform Power Management

Each CPU in a data center requires a large amount of support hardware. This support hardware, contained within the server chassis, is generally referred to as the *platform*. Over the years, more and more of the platform has been integrated into the CPU, such as memory controllers and PCIe connectivity. However, a large portion of the overall power in the data center is still consumed by the support infrastructure outside of the CPUs and memory. Storage (drives), networking, power delivery, and cooling all can contribute a significant amount to the overall cost of a data center. Some of these components (like the fans) have sophisticated algorithms that attempt to manage their power consumption, whereas others (like drives) tend to employ minimal power management techniques.

Platform Overview

A *platform* is conceptually everything (including the CPU) required for a CPU to operate. It includes the power delivery (which converts electricity from the power grid into something usable by the different platform components), cooling (fans, heat sinks, etc.), as well as the memory, drives, and networking that are connected to the CPU sockets.

Common Platform Components

A single platform is commonly referred to as a *node*, which generally incorporates from one to eight CPUs that are connected with coherency.[1] A wide range of platform designs are possible and available. However, some standard building blocks go into just about any platform design (see Table 4-1). This chapter investigates some of the power management characteristics of these various platform components.

[1] *Coherency* is a mechanism that allows different software threads running on different CPUs to share a large set of physical memory without requiring software management.

www.dbooks.org

Table 4-1. *Common Platform Components*

Component	Description
CPU	These processors provide the computation and execution of user workloads. See Chapter 2.
Memory	Memory provides temporary storage for data being used by the CPUs. See Chapter 3.
Storage	Storage (drives) provides bulk storage of data. SAS (serial attached SCSI) and SATA (Serial ATA) are two common protocols for connecting drives to a storage controller.
Networking	Networking provides for communication between multiple nodes. Ethernet and InfiniBand (IB) are common networking interfaces. NICs (network interface cards) provide the connectivity between the CPU and the Ethernet/IB network.
Power delivery	Different components in the system require different voltages and types of current (AC/DC). VRs (voltage regulators) are DC to DC converters that take an input voltage and step it down to a lower operating voltage. PSUs (power supplies) take AC current and convert it to DC.
Cooling	When servers consume power, it is turned into heat. Fans and other cooling devices are used to extract that heat from the platform to maintain a safe operating temperature.

A wide range of platform designs are used in the industry. Some designs provide large amounts of data storage and connect a large number of drives. Others may be completely driveless and use the network to bring data into the node. Figure 4-1 provides an example of one potential platform node with two CPU sockets.

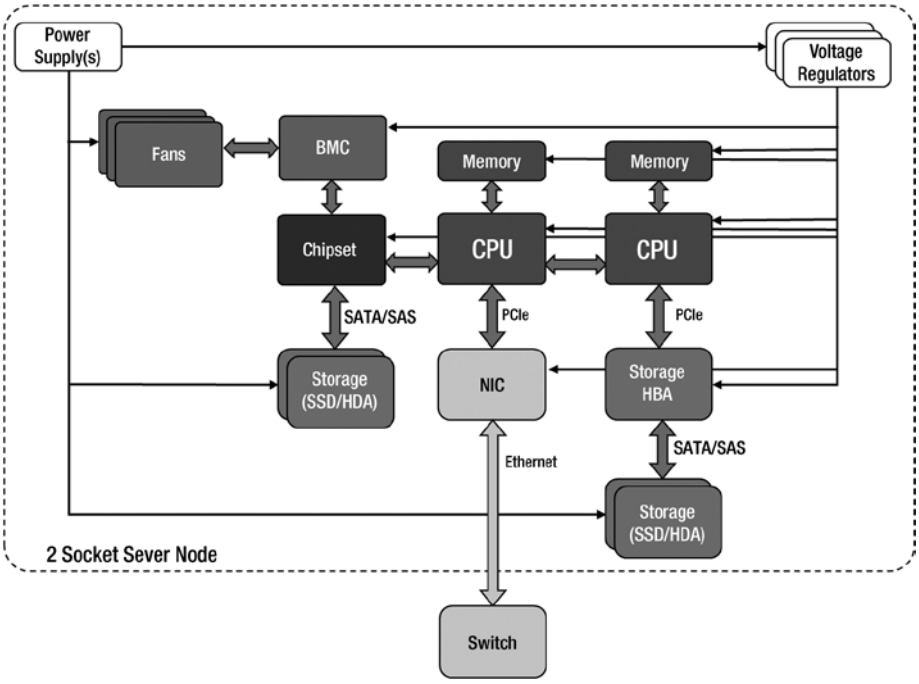


Figure 4-1. *Two socket platform node example*

Integration

As time has progressed, more pieces of the platform have been integrated into fewer discrete chips. This can save cost and power and even improve performance in some cases.

CPU Integration

For many years, the sole role of the CPU die was to provide one (or a couple of) cores and a supporting cache hierarchy. These were then connected to some system bus (front-side bus (FSB) on Intel systems), which then connected them to a chipset. This chipset provided a memory controller and PCI connectivity for devices like drive and network controllers. These busses consumed power, limit bandwidth, and increased latencies. As a result, more and more of the chipset began to get integrated into the CPU itself, both to reduce platform power and to increase performance. Table 4-2 provides an overview of some of the key integration milestones over Xeon processor generations.

www.dbooks.org

Table 4-2. *Integration over Intel EP CPU Generations*

Generation	Integration
Nehalem	Memory controllers
Jasper Forest (Nehalem Derivative)	PCIe 2.0 (up to 16 lanes)
Sandy Bridge	PCIe 3.0 (up to 40 lanes) that can share L3 cache with cores
Haswell	Voltage regulators

Although power can be saved (fewer platform busses and I/Os driving them) and performance can be improved (on-die busses provide higher bandwidth and lower latency compared to off-chip interconnects), this integration is not free. The area of the CPU must increase to accommodate the additional components. CPU packages may need to accommodate more pins (which increases cost). This integration also moves power that was previously consumed out in the platform into a much closer physical location to the traditional CPU components. This either requires that more power (and cooling) be provided to the CPU or that less power be made available to the cores.

CPUs are typically built on the latest manufacturing process technology that provides the best power efficiency. Other devices in the platform are usually manufactured on older technologies. When they are integrated into the CPU, these capabilities get an immediate upgrade in power efficiency due to the process technology improvement.

Chipset Integration

The CPU absorbed the memory controller and some of the PCIe connectivity away from the chipset in the Nehalem and Sandy Bridge generations. However, the chipset has started integrating other components of the platform. Storage and network controllers are now standard on server chipsets. PCIe is still provided, although it is generally lower performance than the CPU links and is focused on low-bandwidth connectivity. Chipsets are discussed in more detail later in this chapter.

Microservers and Server SoCs

Server system on a chip (SoC) components are becoming more and more prevalent. In these designs, the chipset and CPU are integrated together into a single die or as a multi-chip package (MCP). The primary goal here is to reduce the costs of deploying a single CPU node. The concept of a microserver is where you target these lower-cost devices in mass quantities in a data center to provide adequate performance at reduced costs. Although microservers have received significant press in recent years, deploying these lower-cost, power efficient, highly integrated devices into embedded markets is arguably even more interesting.

Platform Manageability

Running a large data center requires capabilities for monitoring and managing the various components that go into a data center. Controlling fans, rebooting nodes that have crashed, monitoring power, and many other tasks are all critical to managing a typical data center. These concepts will be discussed in Chapters 5 and 9. Rather than having software running on the CPU cores to provide these capabilities, many server platforms have traditionally deployed dedicated management chips. These are commonly called baseboard management controllers (BMCs).

Server BMCs are OEM-proprietary devices with a small microcontroller at their heart. They have tentacles throughout the platform in order to monitor and control the various subsystems. Platform Environment Control Interface (PECI) is a standard used for interactions between BMCs and CPUs. System Management Bus (SMBus) protocol is also commonly used for providing telemetry information from platform devices (power supplies, etc.) to the BMC. Intelligent Platform Management Interface (IPMI) is an interface used for software to interact with the BMC for extracting the wealth of information of which the BMC is aware (see Chapter 7 for examples). A single platform with *N* coherent CPU sockets is generally paired with a single BMC, but this is not strictly required.

BMCs themselves do not consume a significant amount of power but can have a notable impact on the overall power draw of the system since they control the fans associated with a given platform node. Thermal management is discussed later in this chapter. Servers without BMCs have been investigated in order to reduce power consumption and save on integration costs, but thus far, such designs have not taken off.

CPU Sockets

Modern CPU nodes can support varied numbers of CPU sockets. Uni-processor (UP) and dual-processor (DP) servers make up the bulk of the server processor nodes sold today.

Multi-processor (MP) nodes commonly consist of four or eight processors, but other topologies are also possible. MP platforms have a higher procurement cost associated with them, and are frequently used in situations where large single-node performance or memory capacity is required. By moving to a larger number of CPU processors per node, the cost of some of the platform components can be amortized. For example, if each node requires a boot SSD and a network connection, one can potentially reduce the number of required SSDs and network connections by two times by going from a UP to a DP platform.

■ **Note** Due to the large procurement costs and usage models associated with MP systems, power efficiency and power savings are typically a lower priority for end users.

---

www.dbooks.org

DP platforms provide an excellent cost/performance sweet spot. MP platforms have typically demanded a higher overall price per CPU, while UP platforms are not as effective at amortizing other platform costs (power and procurement). DP platforms also exhibit strong performance scaling for many workloads.

UP server systems have traditionally been relegated to situations that simply did not demand the performance of a DP or MP system. Rather than being deployed in a data center, they have been used in other lower-end server appliances such as small business NAS (network-attached storage). As single node performance continues to increase, UP systems cost amortization is improving. If a DP system requires two network connections in order to provide sufficient data to saturate the capabilities of the cores, then there is no additional savings by scaling to two sockets. Server SoCs (like microservers) that incorporate capabilities like networking also help reduce the power and procurement amortization benefits of multi-socket systems.

Platforms that directly connect two to eight processors coherently to each other are said to be *glueless*. A variety of glueless topologies have been developed over time. Figure 4-2 shows some examples from recent processor generations from Intel. Note that in each of these examples, every socket is either one or two "hops" from each other socket on the platform. It is possible to connect even more processors in a coherent network, but this generally requires special hardware (or glue) called node controllers. If all the processors are directly connected to each other through point to point links, the platform is said to be *fully connected*. Fully connected platforms generally have lower latencies, higher bandwidth, and better performance scaling than platforms that are not. There is a small power cost for the additional connectivity, but the return on investment (performance) is well worth the cost for most usage models.

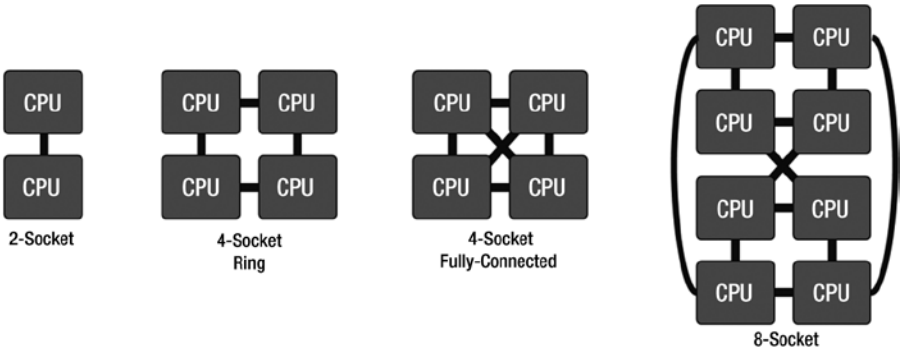


Figure 4-2. *Example glueless coherent platform topologies*

Node Controllers

Although the majority of platforms limit the number of coherent CPUs to a maximum of eight, it is possible to build much larger coherent systems using *node controllers (xNC)*. Node controllers are generally discrete chips that connect one, two, or four CPUs out to other node controllers through a proprietary fabric (see Figure 4-3). These systems are frequently used for building supercomputers and can connect hundreds of processors and thousands of cores into a single coherent domain running a single operating system.

Note that it is also possible to build large supercomputers without node controllers by connecting a large number of nodes non-coherently through a network. The differences between these designs are beyond the scope of this book.

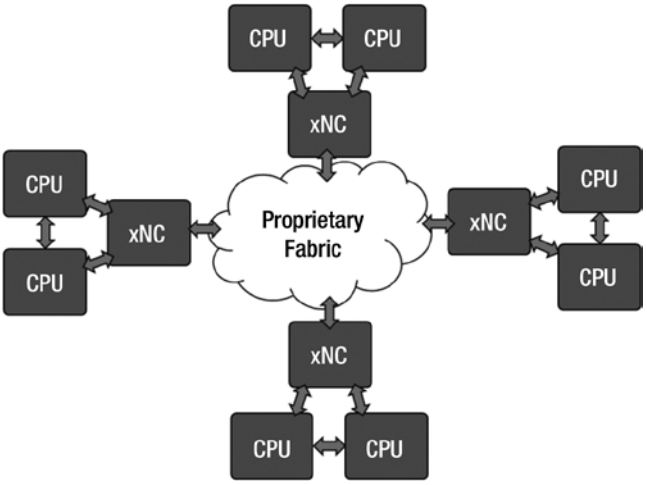


Figure 4-3. *Node controller example*

Memory Risers and Memory Buffer Chips

Certain high-end servers demand very large memory capacities. Databases are the most common example. Each CPU socket is generally limited in the amount of DDR memory to which it can directly attach. The number of DDR channels on a socket is constrained by packaging and die costs. The number of DIMMs on a channel is limited by electrical loading constraints. LR-DIMMs attempt to address some of these issues but can only go so far. In order to expand memory past the constraints imposed by the CPU socket, memory risers and memory buffer chips have been used on some high-end servers. Rather than connecting the CPU directly to memory, the CPU communicates with a discrete chip in the platform that is then able to communicate to the actual DDR memory. In these platforms, the memory is connected on separate riser cards, where a set of DIMMs is connected to a card, and then that card is connected into the motherboard. There have been various flavors of these technologies over the years. Intel has historically productized a memory buffer technology as part of its EX platforms (called Scalable Memory Buffer [SMB]), and other OEMs have deployed their own proprietary technologies to provide similar capabilities. These buffer chips do consume measurable power (usually a few watts), but they tend to be dwarfed by other power in such platforms (including the memory that they provide connectivity to).

www.dbooks.org

Server Chipsets

Many server platforms employ discrete chipsets that are connected to the CPUs. These devices provide key legacy capabilities required for booting the platform, capabilities for manageability, and also integration of many features that otherwise would require discrete controllers (such as storage, network, and USB controllers). Some SoCs integrate the chipset functionality into the package with an MCP, while others (like Avoton) integrate the entire chipset into the same die as the CPU. The discrete chipsets used in many Xeon server designs at Intel are called a PCH (Platform Controller Hub). The PCH attaches to the CPU via a proprietary DMI (Direct Media Interface) link, and provides boot, manageability, and I/O services to the platform.

The PCH has been the south bridge of the two-chip Xeon Intel Architecture since the Nehalem/Tylersburg generation and is a companion to the CPU. This architecture succeeds the Intel Hub Architecture, which was a three-chip solution. Successive generations of PCH have advanced the I/O capability of IA platforms, with Gen2 PCIe, Gen3 SATA, and Gen3 USB now available on Wellsburg. A microcontroller-based power management controller (PMC) and a Management Engine (ME) were added to the PCH to support traditional power management features, along with several extended features.

The chipset serves a variety of purposes in the platform. Table 4-3 provides a high-level summary of some of the key capabilities. Figure 4-4 shows an example block diagram of such a system. Table 4-4 enumerates some of the integrated functionality of modern PCHs.

Table 4-3. *PCH High-Level Capabilities*

Capability	Description
High-performance I/O connectivity	This includes PCIe, storage (SATA and/or SAS), networking, etc. These capabilities are only available when the CPU is active and the system is in the S0 state.
Wake/boot	The PCH both detects wake events (like Wake on LAN) and sequences the platform to transition in and out of platform power states. The PCH also provides access to flash memory for BIOS.
Manageability	This provides interfaces for the data center to monitor and manage the node, such as reading temperatures.
Real-time clock (RTC)	This maintains the system clock that tracks clock time. If you unplug a desktop from the wall, your time and date is not lost since it is maintained on the RTC. The same capability exists in server PCHs.
Legacy I/O connectivity	This provides connectivity to low-performance platform connectivity that is generally required for system operation.

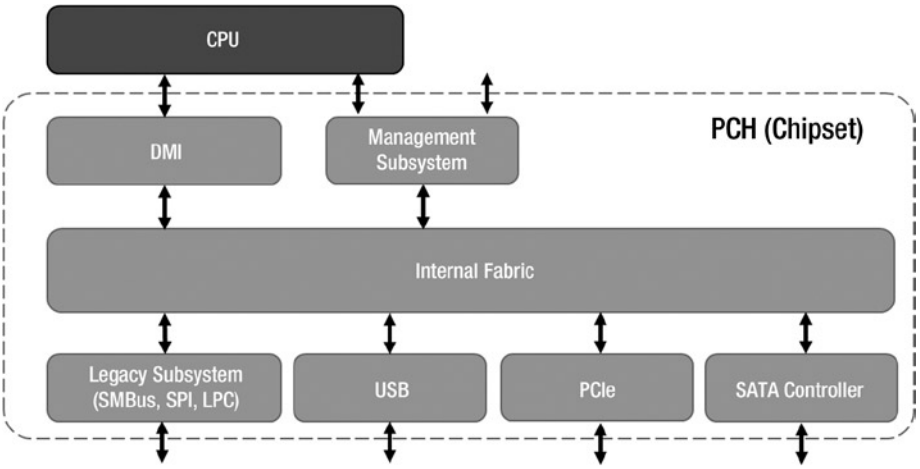


Figure 4-4. *A typical server PCH architecture block diagram*

Table 4-4. *Primary PCH Components*

Component	Description
On-die fabric	Interconnects exist on the PCH that are commonly called *on-die* or *on-chip* fabrics. These interconnects are conceptually similar to those in CPUs. These are not to be confused with fabrics that connect multiple CPUs together at the data center level.
DMI	DMI provides a mechanism to connect the PCH and components connected downstream from the PCH to the CPU. It operates very similar to PCIe.
PCIe	PCIe connectivity can be incorporated both into the CPU and the PCH and shares the same basic power management capabilities. The PCH is useful for high fanout, low bandwidth connectivity.
SATA	Storage connectivity is included on some PCHs and is discussed in the "Storage" section of this chapter.
USB	USB is primarily targeted and consumer usage models but is also present in servers (particularly for debug usage models). It is discussed later in this chapter.
Ethernet	Ethernet integration is also incorporated into the PCH. Networking power management is discussed in the "Networking" section of this chapter.

(*continued*)

www.dbooks.org

Table 4-4. (*continued*)

Component	Description
SPI/LPC (legacy I/O)	Serial Peripheral Interface (SPI) and Low Pin Count Interface (LPC) provide connection points to platform boot devices that contain the BIOS/UEFI image, as well as firmware for other PCH components (e.g., ME, Ethernet).
ME	The Management Engine on the PCH provides platform management services, key management, and cryptographic services.
SMBus	SMBus provides a legacy mechanism for communication with platform peripherals for system and power management–related tasks.

The PCH has traditionally been the component that provides access to various high-speed I/Os (SATA, Ethernet, PCIe, USB), although these capabilities are increasingly being integrated into the CPU to create SoC components. The SATA/PCIe/Ethernet/USB interfaces provide access to external communication, including disk/solid-state storage, networking, USB ports, and manageability. Moving data with higher performance between two devices in a platform consumes non-trivial power, and integration is an effective way to significantly improve overall platform power consumption.

Internally, the PCH architecture is constructed with a mix of analog and digital components. Similar to the CPU uncore, analog design is used for designing the off-chip communication (e.g., PCIe/SATA/USB physical interface) and on-die memory (SRAM), while the bulk of the remaining system is built out of synchronous digital logic. Unlike traditional server CPUs, a large percentage of the chipset power is consumed by analog I/O circuitry (commonly called physical layer or PHY) and not the digital logic.

PCH and Platform Power Management

The PCH orchestrates many of the platform power states introduced in Chapter 2. In addition to this task, it is responsible for managing its own power states. Table 4-5 provides an overview of the power management states in which the PCH participates.

Table 4-5. *System States Supported by Server PCH*

State	Description
C-states	*Low-power states for PCH I/Os* See Chapter 2 for details on CPU C-states. PCH does not support all the traditional CPU C-states but places its I/Os in low-power states when the CPU is not active. • C0 is an active state when the PCH logic and I/O are functional. • Cx is a clock/power-gated state, during which PCH I/Os are transitioned to a lower power-managed state.
S-states	*Turning off the CPU package (sleep state)* See Chapter 2 for details on S-states. The PCH I/Os are turned off (except S0), but the PCH core logic remains active in all S-states. PCH can wake the CPU up from S3/S4/S5 states based on platform signaling. Waking from an S3 state takes seconds, whereas waking from S5 requires a full system boot and can take multiple minutes.
M-states	*Turning off the Management Engine* These states are related to the Management Engine. • M0: Active state, when platform is in S0 state. • M3: Active state, when platform is in S3/S4/S5 state, used for out-of-band platform management and diagnostics. • MOff: Management engine is turned off in S*x*.
G-states	*Global states* See Chapter 2 for details. The PCH is active in G0 to G2 and is only off in the G3 state (mechanical off).

Since the PCH controls the platform rails and clocks, it needs to remain powered on even in states where the CPU is powered off (G- and S-states). This is accomplished by using platform power rails that are successively powered off depending on the system state. The PCH provides a number of high-level capabilities that are successively disabled at lower power states. Table 4-4 provides an overview of these capabilities, and Table 4-6 shows how the capabilities are disabled in each power state.

www.dbooks.org

Table 4-6. *Example PCH Power States and Capabilities*

G-State	M-State	S-States	Power Rails	Manageability	Wake Capability
G0	M0	S0	All available	Available	N/A (awake)
G0	M3	S3/S4/S5	Wake + Manageability + RTC	Available	Yes
G2	MOff	S4/S5	Wake + RTC	Disabled	Yes
G3	MOff	S4/S5	RTC	Disabled	None

Systems autonomously transition out of the G3 state and into the G2 state when power is supplied to the platform. From there, various wake events can be used to transition the state into a higher power operational mode as needed. As such, the G3 state is hidden from the user.

PCH Power Management

The PCH consumes a small percentage of a node's overall power. Because the bulk of the power consumed by the PCH exists in the I/O PHYs, the typical power consumed under load is very dependent on the number of connected devices. The TDP power of Patsburg (the chipset used with Sandy Bridge and Ivy Bridge E5 processors) was 8 W to 12 W when all of the high-speed I/O ports were connected (fully populated). Wellsburg (paired with Haswell E5) consumed a TDP of 7 W when fully populated. Notable power can be saved if certain I/Os are not populated. Table 4-7 provides an overview of four different usage configurations of the PCH and the corresponding TDP power for those configurations.

Table 4-7. *PCH TDP Power (W) Consumption with Various I/O Port Configurations*

	Workstation	Server	Low Power	Boot-Only
USB2 Ports	14	6	2 (detection)	0
USB3 Ports	4	4	1 (detection)	0
SATA3 Ports	8	5	2	0
SATA2 Ports	2	1	1	0
PCIe Lanes	8	4	2	0
TDP (W)	6.5	5	3.2	1

The Wellsburg PCH, which launched with the Haswell Server CPU, is built on a low-leakage process and does not implement techniques like voltage-frequency scaling or power gating to reduce the power consumed at runtime. Turbo is not available. In order to save power, clock gating is performed on logic features that are disabled or not currently in use. Since the PCH is I/O dominated, a sizable portion of the power is

consumed by the circuits that provide the physical interface to the platform. The voltage of these interfaces is generally static (as defined by industry standard specifications). Several power management states are defined for the links to opportunistically reduce power based on the operating state, as described in the following subsections.

If an entire section of logic is not being used, then the PLL (phase-locked loop) that drives that logic can be powered down. For example, if a user is building a compute node that lacks any local drives, the storage subsystem in the PCH can be completely powered down.

PCIe in Chipsets

Prior to Sandy Bridge and Jasper Forest, chipsets provided the PCIe connectivity in the platform. When Sandy Bridge integrated PCIe into the CPU, the chipsets continued to provide this capability. Today in platforms with discrete PCH devices, PCIe connectivity is offered on both the CPU die and the PCH. PCIe in the PCH provides the same power-saving capabilities that are described in Chapter 3 (L1, DLW).

PCIe on the CPU provides high performance and (relatively) low latency connectivity at the expense of limitations in the fanout (devices smaller than x4 consume four lanes). The PCH, on the other hand, provides lower bandwidth and longer latencies, but can be bifurcated down to x1 making it an excellent choice for low bandwidth devices.

PCH Thermal Management

The PCH contains thermal sensors in order to monitor the temperature and help guarantee that the PCH will not get to a dangerous temperature where reduced reliability or damage could occur. The PCH may throttle itself to stay under a target temperature or even initiate an immediate shutdown if temperature exceeds a catastrophic threshold. Like CPUs, PCHs are spec'd with a TDP rating that is used to design the thermal solution and an ICCMAX rating that is used to size the voltage regulators to power the voltage rails. They also contain similar thermal protection mechanisms such as shutting down the platform when catastrophic temperatures are detected. Platform thermal management is discussed in detail later in the chapter.

Networking

Network interfaces—both the local LAN adapter as well as network infrastructure devices—are the gateway for the server platform to the rest of the world. Network activity demonstrates unpredictable distribution of packet arrival times at multiple scales. As a side effect, the network interfaces are never fully powered down. LAN adapters contribute ~5–10 W to the overall platform power. This power is not one of the primary power contributors in typical server platforms that deploy high-power CPUs and large amounts of memory. Although the LAN adapters themselves do not directly contribute a significant percentage of the platform power, their behavior and configuration can have a large impact on the power consumption of the CPU (and thus the platform).

www.dbooks.org

■ **Note** Although network cards do not themselves contribute a significant percentage of the platform power consumption, their configuration and behavior can have significant impacts on CPU power (and thus platform power).

In typical usage, LAN component power is driven by five main factors. Table 4-8 provides a high-level summary of these factors (which are discussed in detail in the following pages.).

Table 4-8. *Primary Factors in LAN Controller Power*

Factor	Description
Ambient temperature	LAN devices have traditionally been manufactured with high-leakage process technologies, resulting in a significant power increase at higher temperatures.
Attached media	The type of connection (fiber optic, copper cable, etc.) can have a moderate impact on the power consumption.
Configured speed	LAN controllers can be configured by software to run at lower frequencies. This can save notable power.
Power management features	Various power management options are available that can trade off performance (latency) to save power.
Bandwidth	Packets per second have the biggest impact on NIC power (not raw bandwidth). However, on recent high-performance networking devices, there is not significant sensitivity to bandwidth.

Ambient Temperature, TDP, and Thermal Management

Many LAN vendors quote typical power numbers in their datasheets. However, there are no industry conventions as to what typical usage is, though many assume 25°C for ambient air temperature, and nominal voltage. An increase in temperature from 25°C to 70°C can increase the component power by 50% to 100% solely due to leakage (which itself is a function of the silicon process used to produce the device). As LAN controllers transition to lower leakage processes or are integrated into low-leakage SoC designs, the sensitivity to temperature will decrease.

Similar to CPU designs, the maximum quoted power of the LAN controller is measured assuming worst-case conditions, including high temperatures (~70°C ambient). The server platform thermal management—such as fan size and speed—is designed to cool to this maximum component thermal design point (TDP). LAN controllers are typically designed assuming passive cooling, and it is also common for these devices to exist in areas of limited airflow. Active cooling—such as fans—is discouraged because of server platform reliability concerns. The net result is, regardless of the functionality or media provided,

the server LAN component TDP must be 10 W or less (unless special design provisions are made at the platform level for additional fan cooling). Tables 4-9, 4-10, and 4-11 show some historical information about Intel LAN adapter TDP power.

Table 4-9. *Historical TDP Power of Single-Port 1 Gbps Intel LAN Adapters*

Year	Device	Ports/Speed	TDP (W)	TDP (W) / Gbps
2001	Intel 82544EI PCI-X	1x 1 Gbps	1.5 W	1.5 W
2004	Intel 82541 PCI	1x 1 Gbps	1.0 W	1.0 W
2005	Intel 82573 PCIe	1x 1 Gbps	1.3 W	1.3 W
2008	Intel 82574 PCI	1x 1 Gbps	0.7 W	0.7 W
2012	Intel I210 PCIe	1x 1 Gbps	0.7 W	0.7 W

Table 4-10. *Historical TDP Power of Multi-Port 1 GBps Intel LAN Adapters*

Year	Device	Ports/Speed	TDP (W)	TDP (W) / Gbps
2005	Intel 82571 PCIe	2x 1 Gbps	3.4 W	1.7 W
2009	Intel 82576 PCIe	2x 1 Gbps	2.8 W	1.4 W
2010	Intel 82580 PCIe	4x 1 Gbps	3.5 W	0.9 W
2011	Intel I350 PCIe	2x 1 Gbps	2.8 W	1.4 W
2011	Intel I350 PCIe	4x 1 Gbps	4.0 W	1.0 W

Table 4-11. *Historical TDP Power of 10 GBps Intel LAN Adapters*

Year	Device	Ports/Speed	TDP (W)	TDP (W) / Gbps
2001	Intel 82597 PCI-X	1x 10 Gbps	9.0 W	0.9 W
2007	Intel 82598 PCIe	2x 10 Gbps	6.5 W	0.3 W
2011	Intel 82599 PCIe	2x 10 Gbps	6.2 W	0.3 W
2012	Intel X540 PCIe w/ 10GBASE-T Phy[2]	2x 10 Gbps	12.5 W	0.6 W
2014	Intel X710 PCIe	4x 10 Gbps	7.0 W	0.17 W

[2]This device includes a 10GBASE-T attached media, increasing the TDP power. The other controllers listed must be paired with a separate attached media.

www.dbooks.org

■ **Note** Typical power for NIC cards is well below their TDP specifications. NICs frequently operate at lower temperatures than their specifications, saving significant leakage power.

In typical usage, the LAN component does not operate at TDP. Some LAN devices include thermal sensor diodes, as well as management interfaces, to enable other platform components to query the component thermal state and adjust fan speed. In practice, many of these platform methods require additional calibration of the thermal sensors which, if not done, may limit the effectiveness of the fan speed algorithms.

Attached Media

Most LAN adapters can be paired with a variety of different interconnect types that provide the actual connectivity between the LAN adapter and network switches. These are called *attached media*.

Server LAN implementations have a greater variety of media types than those found on client systems. Whereas most equate Ethernet to the pervasive RJ-45 connector and 10BASE-T (10 Mbps), 100BASE-TX (Fast Ethernet, or 100 Mbps) and 1000BASE-T (1 Gbps Gigabit Ethernet), server platforms have employed several media types as summarized in Table 4-12.

Table 4-12. *Types of Attached Media*

Type	Max Distance	Power	Latency
Multi-mode short reach (SR) fiber optic	~400 m	~1 W	Slight increase
Single-mode long read (LR) fiber optic	~10 km	~1 W	Slight increase
KX/KX4/KR Backplane (copper)	Server backplane	100s of mW	Best
Direct Attach (DA)	3–10 m	100s of mW	Best
BASE-T	100+ m	2–3 W	Adds ~1 microsecond

■ **Note** Cost and distances are generally the deciding factors in attached media selection. Latency is important to a subset of customers.

Each of these media solutions have tradeoffs between cable cost, power, distance, and even propagation velocity (fiber is slightly slower than copper-based connections). Because of this diversity, many server LAN connections are shipped with an SFP or SFP+ cage, which accepts various media type pluggable modules.

LAN Power Management Features

A number of common features are used for reducing power of both the LAN devices and the CPU. In addition to these, higher-end server LAN adapters implement multiple queues and methods to balance network traffic across multiple CPU cores. As a result, CPU cores can operate at a reduced frequency and save power.

Media Speed

Some media types—such as BASE-T and backplane—support establishing a link at lower media rates than the maximum possible—such as a 1 Gbps adapter linked at 100 Mbps. Lowering the established link rate often reduces the component power, sometimes by as much as 50%. As the link speed drops, the internal synchronized media clock lowers in frequency, leading to a lower dynamic power. Another effect relates to effective packet rates, since LAN component power varies more as a function of packet rate than packet size. For each packet, the LAN controller performs various lookups on the packet headers. Reducing the media rate reduces the packet rates as well, again leading to lower dynamic power.

In practice, changing media speed is not applicable for most server usage models. The transition latency is slow, and the reduced speed results in significant peak throughput reductions and the potential for increased latencies. Although this can save notable power from the perspective of the LAN controller, it is generally not as significant as a percentage of the overall platform power.

Energy Efficient Ethernet

BASE-T and backplane media also support Energy Efficient Ethernet[3] (EEE). This is frequently called *triple-E* for short. EEE devices enter into a low power mode during idle periods, periodically sending idle sequences to keep the link active and sending a wakeup symbol to the peer when the link needs to be reactivated. Depending on the media, the link transitions from idle to active are less than about 16 microseconds. BASE-T devices can reduce their PHY idle power as much 400 mW with 1000BASE-T, and by 2 W with 10GBASE-T.

EEE is managed by the NIC driver and can be controlled at runtime. It is generally enabled by default. The latency cost of EEE is not noticeable in many usage models, but the power savings is also not particularly significant. Latency sensitive users may want to attempt to disable this capability.

[3]EEE is decribed in detail in IEEE Std 802.3az-2010.

www.dbooks.org

Wake on LAN

Wake on LAN (WoL) is a common feature available on server LAN adapters. It is not a power-savings feature as much as a mechanism to wake the system from an S-state.

If the platform supports suspend or wake from soft-off modes, WoL allows remote administrators a simple method to remotely activate a server platform. Often, the LAN interface will reduce link speed automatically when entering this state to minimize power and await receipt of a wake pattern. A common pattern used is the *Magic Packet* pattern. Upon receipt, the LAN controller asserts a signal to the platform to bring the system out of the low-power state.

Active State Power Management (ASPM)

Discrete LAN controllers are connected using PCIe. As such, PCIe Active State Power Management is available to manage power on LAN controllers. PCIe power management is discussed in Chapter 3.

NIC ASPM L1 is frequently disabled in server deployments. This can frequently be performed in the system BIOS. The latency implications are frequently not worth the low amount of power savings. One common issue with ASPM L1 is that it blocks communication between a driver (running on the CPU) and the NIC device. When communication is required, the core is then stalled. This ends up wasting CPU power, which eats into the already small savings from L1.

Interrupt Moderation

Interrupt moderation is another common feature of LAN controllers. It limits the rate at which interrupt signals are delivered to the host CPU. This often reduces the CPU utilization with little to no observable impact to bandwidth. Interrupt moderation has little impact on NIC power, but the decreased CPU utilizations can significantly improve CPU power consumption. It can also make additional CPU cycles available for other processes, improving the throughput of the node. By rate limiting interrupts, the CPU is notified less often, resulting in an increase in latency and response time. The amount of latency impact can be tuned inside the NIC driver and is commonly configured to levels on the order of 100–200 microseconds. This feature is typically enabled by default, and can be disabled (or configured) inside the NIC driver configuration.

Interrupt moderation can have a significant impact on power and latency in systems, and is frequently overlooked. Tuning this feature should be a priority for anyone who is concerned about latency and response times.

DMA coalescing is a related feature that attempts to queue up data transfers inside the NIC and burst them into the CPU. The intention of this feature was to allow the CPU to get into a low-power idle state between bursts of activity. In practice this feature has shown minimal effectiveness in server environments while also significantly increasing network latencies. It is not enabled by default.

USB

USB connectivity is provided by server PCHs. Many large-scale data centers do not connect devices to USB under normal operation, but it is common for USB ports to be included on those platforms. “Crash cart” support is a common usage model, where USB is periodically used to connect a keyboard/mouse for local debug, or to connect a USB drive for similar purposes. USB can also be used in some low-end storage systems for connecting USB storage. Power management of USB can be very effective at saving power at minimal to no power cost due to these limited usage models.

Link Power States

The initial USB power management capabilities were very coarse grained. A *suspend/resume* scheme provided two levels—effectively “on” and “off.” These take milliseconds for transitions, making them inadequate for many power-efficiency usage models. USB devices are common in consumer usage models where achieving very low idle power is critical to achieving long battery life. As a result, USB has been a focus for power optimization in these environments. Much of these capabilities are unnecessary in server usage models.

USB 2.0 originally only supported these two levels but later added support for L-states that complemented the suspend state. On USB 2.0, the state of the link is tied to the power state of the device. These states are summarized in Table 4-13. Suspend can still be used for states that have no latency sensitivity (such as S3/S4).

Table 4-13. *USB 2.0 Power States*

State	Name	Link Savings	Device Savings	Exit Latency
L0	On	--	--	--
L1	Sleep	~100 mW	Device-specific	microseconds
L2	Suspend	~125 mW	Device draws almost no power	milliseconds
L3	Off/Disconnected	~140 mW	Device powered down	milliseconds

Note: Power savings are design dependent. These numbers provide a reference point.

On USB 3.0, the device power states were decoupled from the link power states. U-states were defined that control the power state of the link only. Table 4-14 provides an overview of the four USB 3.0 link power states.

www.dbooks.org

Table 4-14. *USB 3.0 Link Power States*

State	Name	Link Savings	Exit Latency
U0	Link active	--	--
U1	Link down	~100 mW per lane	Microseconds
U2	Link down	~125 mW per lane	Milliseconds
U3	Link off	~140 mW per lane	Milliseconds

Note: Power savings are design dependent. These numbers provide a reference point.

Link Frequency/Voltage

USB has gone through three generations. Each generation has a single frequency at which the device runs (see Table 4-15). Multiple voltage/frequency points are not supported. Newer generation devices support (by rule) backward compatibility to the prior generation frequencies.

Table 4-15. *USB Generations*

Generation	Frequency	Duplex	Theoretical Bandwidth
USB 1.*x*	12 MHz	Half	1.5 MB/s
USB 2.0	480 MHz	Half	35 MB/s
USB 3.0	5 GHz	Full	500 MB/s (per direction)

USB 3.0 moved to a full-duplex design, effectively providing separate communication channels for both directions, increasing the peak throughput when data are transferred in both directions simultaneously. This required the addition of two more differential pairs, and is similar to a single lane of PCIe or a SATA connection.

Storage

Many modern data centers deploy storage in a variety of different ways. Some compute nodes have no local storage and depend entirely on the network to provide access to remote storage. It is also common to see compute nodes with a single drive (commonly an SSD) that provides for high-performance local storage. Other nodes can be targeted for storage and can provide access to a large number of drives. These nodes are connected to compute servers through high-performance interconnects to provide large pools of shared storage.

Traditionally, drives have been connected through PCIe-based controllers. Two standard interfaces exist for these controllers: SATA and SAS. Serial Advanced Technology Attachment (SATA) is the lower cost of the two, but it also provides lower peak performance. SATA can be used both in consumer and server usage models. Serial attached SCSI (SAS) is generally more expensive and higher performance and is targeted at enterprise usage models. SATA drives can be connected to a SAS infrastructure, but SAS drives cannot be connected to a SATA controller. SATA and SAS support both SSDs (solid state drives) and HDDs (hard disk drives). In addition to providing higher peak performance, SAS provides the ability to connect a large number of drives to a single controller, making it popular in very high capacity deployments.

In recent years, SSDs have begun to be directly connected on PCIe. Non-Volatile Memory Express (NVMe) is a specification for performing this direct connection. NVMe provides lower latency and higher performance than SAS and SATA. This is particularly well-suited for high performance compute nodes that require local storage. NVMe SSDs exhibit similar power characteristics to SATA and SAS SSDs. Their potential for higher performance also translates into higher power consumption.

Storage power consumption is generally not a significant component of the overall node power in traditional compute servers. However, the power consumption of the drives on a storage node can dwarf the other components on the node. Storage power is also more significant in low-power, low-performance servers where the drive power is not amortized across a high-power CPU node.

Storage Servers and Power Management

In a typical storage server the CPU complex manages tens to thousands of drives. Storage servers can use a mix of SSDs and HDDs, and the mix is determined by the performance needs. SSDs have higher procurement costs but provide improved performance. A significant amount of power in storage servers is consumed by the drives. Cooling a dense storage complex can also consume non-trivial power.

It is increasingly critical to manage the power consumed by the storage devices, without adversely affecting performance. Various power management schemes can be used depending on the performance requirements of the application. In a cold storage system where a massive amount of data is maintained but accessed rarely with low performance and latency requirements, aggressive power management can be used to reduce costs. On the other hand, limited power savings is used in performance- and latency-critical storage deployments. Aggressive power savings in such environments can be detrimental to performance but can also reduce overall data center power efficiency by forcing compute nodes to wait longer for data (wasting power in the process).

Power savings opportunities exist both within the drives and in the communication layer between the drives and their controllers. Storage power management schemes have been developed for both server usage models as well as consumer usage models. In consumer usage models, very low idle power is critical for battery life, and capabilities have been developed to address these concerns. These same capabilities may be available in the server space but can provide poor tradeoffs. A few hundred milliwatts of power savings is commonly a poor tradeoff if it could result in milliseconds of response time increase.

www.dbooks.org

HDDs and SDDs

SSDs generally provide higher bandwidth and lower latencies than HDDs; however, this has traditionally come at increased power and cost per capacity. Actual power consumption is drive dependent, and can range anywhere from a couple of watts to more than 10 watts.

Traditionally, 3.5" HDDs have been extensively used in data centers and other server applications. They can provide the best cost per GB due to the larger platters. 2.5" HDDs do tend to exhibit lower power draw than 3.5" drives of the same capacity, but this benefit tends to be overshadowed by their overall lower maximum capacity in the same technology generation.

The amount of traffic that an HDD is servicing has minimal impact on the amount of power that is consumed. Rather, the state of the drive (Is it spinning? Are the heads loaded?) is the predominant component of HDD power consumption. SSDs, on the other hand, exhibit significant power dynamic range as a function of bandwidth. The act of reading and writing the cells itself consumes a significant percentage of the drive power. SSDs' power consumption also appears to scale with capacity. This is not because the cells themselves consume significant power, but because peak performance of these higher capacity drives is frequently higher, providing more potential for power consumption.

Power consumption during spin-up of an HDD is often the highest power draw of all of the different operating states of an HDD. It can dwarf the power consumption of normal operation. In storage servers with a large number of HDDs, staggered spin-up can be employed to prevent the excessive power consumption of spin-up, which may result in a power shortage. Staggered spin-up starts one drive at a time, either waiting for the drive to signal that it is ready or waiting a predefined amount of time prior to starting the next drive. Many data center designers are concerned about the *provisioned power* of each node, rack, and so on. This is the amount of power that the system must be designed to provide and is generally less than the sum of the worst-case power of every individual subcomponent in the system. When spin-up is staggered, platform power delivery does not need to be designed for this high-peak power across many drives simultaneously, which means that both cost and the potential for compute density improve. However, this comes at the cost of additional potential latency when drives are spinning down for power efficiency reasons.

SATA and SAS Drive Power Management

Power management of drives can be split into two categories: saving power on the actual drive and saving power on the interconnect (PHY). SAS and SATA have many similarities in their power management methodologies and terminology. Power management of the drive itself is significantly more important than the PHY.

SATA devices (drives) support four power states as shown in Table 4-16. These states can save significant power. The Sleep state is rarely used on servers. You would likely not want to use Standby with HDDs on a compute server when any activity is possible, but careful use is possible on large storage arrays.

Table 4-16. *SATA Device Power Savings*[4]

Action	Description	Receives Commands	HDD	Wake Latency
Working (active)	Normal operation. Fully powered.	Yes	Spun up	N/A
Idle	Active power savings. May take longer to respond to commands.	Yes	Spun up	Milliseconds
Standby	Device still responds to typical commands, but response time may be significant.	Yes	Spun down	<= 30 seconds
Sleep	Device is put to sleep. Must be explicitly woken up.	No	Spun down	<= 30 seconds

SAS power management is conceptually very similar to that of SATA. The PHY supports both Partial and Slumber states with the same characteristics. In the T10 SAS standard, additional states are defined. Although the ATA8-ACS SATA standard only calls for the four states enumerated in Table 4-16, SATA drives may also support similar states to SAS. Table 4-17 provides a summary of these states with ballpark power savings estimates. Note that significant HDD power savings is only possible when significant exit latency costs are accepted. As a result, these power savings modes are typically only deployed after significant idle periods (if at all).

Table 4-17. *SAS/SATA HDD Power Savings Modes*

State	Spinning	Heads	Power Savings	Exit Latency
Active	Full speed	Loaded	Baseline	N/A
Idle A	Full speed	Loaded	~10%	~100 ms
Idle B	Full speed	Unloaded	~20%	~200–400 ms
Idle C	Reduced speed	Unloaded	~50%	Seconds
Standby Y (SAS)	Spun down	Unloaded	~90%	Seconds
Standby Z (SAS) Standby (SATA)	Spun down	Unloaded	~90%	Seconds

[4]ATA8-ACS Standard. www.sata-io.org/sites/default/files/images/SATAPowerManagement_articleFINAL_4-3-12_1.pdf.

www.dbooks.org

For the PHY, both SAS and SATA supports two low-power modes; Partial and Slumber (see Table 4-18). After some predetermined period of inactivity, either the host or the device can signal the PHY to enter its reduced power state. PHY power management has moderately long wakeup latencies, limiting the ability for fine-grained power savings. Slumber has quite long wake latencies, which preclude them from being used in some server usage models. Partial also achieves idle power on the order of ~100 mW, so further power savings at the expense of latency can be counter-productive at the platform level outside of deep idle platform states. DevSleep is predominantly a consumer device power state.

Table 4-18. *PHY Power States*[5]

Action	Wake Latency
Active (SAS)	N/A
PHY Ready (SATA)	
Partial	<10 µs
Slumber	<10 ms
DevSleep (SATA)	~1 s

SSD drives are common in both consumer and enterprise environments. However, these drives have different characteristics and optimization points. Low-power operation and idle power optimization is critical in the consumer space, and the drives have been optimized for those cases. On the other hand, this has been less of a focus in many enterprise drives. Unlike with HDDs, enterprise SSDs may consume only 25% (or less) of their peak read bandwidth power while running in an Active Idle state (and maintaining fast response times). Traditionally, enterprise SSD procurement costs have dwarfed power costs, and users were not likely to deploy SSDs into areas that would be exposed to significant idle periods. As time progresses, the cost per GB of SSDs is decreasing. This will enable SSDs to compete in usage models traditionally reserved for HDDs and is expected to make power management more of a focus.

Frequency/Voltage

SATA and SAS both have evolved over time by increasing frequencies to provide higher bandwidth. Table 4-19 illustrates how these generations have evolved. Newer generation devices support (by rule) backward compatibility to the prior generation frequencies, and the operating voltage has been held constant.

[5] www.sata-io.org/sites/default/files/documents/SATADevSleep-and-RTD3-WP-037-20120102-2_final.pdf

Table 4-19. *SATA and SAS Generations*

SATA Generation	SAS Generation	Link Frequency	Theoretical Peak Bandwidth
SATA 1.*x*	SAS 1.0	1.5 GHz	150 MB/s
SATA 2.*x*	SAS 1.0	3 GHz	300 MB/s
SATA 3.*x*	SAS 2.0	6 GHz	600 MB/s
-	SAS 3.0	12 GHz	1200 MB/s

SATA and SAS traditionally transfer data serially (1 bit of data at a time) and use 8b/10b[6] encoding, which consumes 20% of the data in order to support the high-speed transmission.

80% * (1.5 GHz / 8 bits per byte) = 150 MB/s

SATA 3.2 includes support for PCIe connected devices, which can leverage the parallel nature of PCIe to achieve even higher throughput. It includes the SATA 3 capabilities for traditional SATA connectivity (at the same bandwidth).

NVMe Drive Power Management

NVMe provides power management capabilities that allow the power to be scaled down at the cost of lower throughput and higher latency. Seven different states are defined (numbered 0 to 6) as shown in Table 4-20.

Table 4-20. *NVMe Power States*

State	Operational	Exit Latency	Performance
0	Yes	--	Peak
1 to 4	Yes	Microseconds	Degraded throughput and latency with increasing exit latency
5	No	~50 ms	
6	No	~500 ms	

NVMe allows the host to manage power statically or dynamically to complement autonomous power management performed by the NVMe drives. When power is managed statically, the host predetermines the power allocated to the NVMe drives and sets the NVMe power state of each drive. When the host manages power dynamically, the NVMe power state of each device is updated periodically to accommodate changing performance and power requirements of the host.

[6] *8b/10b* is an encoding scheme that takes 8 bits of data and transfers it using 10 bits. It is used to transmit data over some high-speed interfaces.

www.dbooks.org

■ **Note** NVMe is frequently used in compute servers that demand peak performance. As a result, aggressive power management, particularly with states 5 and 6, may not be a good match.

NVMe is a relatively new technology. Power management of enterprise-class NVMe drives has not been a priority for many users or designs. Many of the initial enterprise offerings do not implement these power management states.

Power Delivery

There are a large number of components within a server platform, and each of them requires power to provide their necessary function. Different components in the system have different requirements for the type of power that they receive. Some need high voltages, others low. Some are very sensitive to operating at a very specific voltage, whereas others are able to tolerate a range of voltages.

Since the type of power provided to a server system is similar to the power you get from your home's wall outlet, the system has many power converters to convert this AC (alternating current) voltage to the many specific DC (direct current) voltages needed by all its components. The conversion of this AC voltage to the required DC voltages consumes power, referred to as *losses* in the converter. The typical measure of these losses in the power converters is expressed as an *efficiency*. Efficiency is expressed as the ratio of the output power to the input power. Since the input power equals the output power plus the power losses of the converter, the efficiency can be expressed as the following equation:

$$Efficiency = \frac{Output\ Power}{InputPower} = \frac{Output\ Power}{Output\ Power + Converter\ Power\ Losses}$$

How efficiently these power converters convert power from higher voltages to the lower voltages that are required by the loads is critical to the overall efficiency of the system. Even in systems with the best converter efficiency, these losses can make up 10%–20% of the power in the system. At low system utilizations, they can contribute an even higher percentage of the power. This section provides an overview of these power converter losses, basics of the different type of power converters used in the system, the various elements of the power conversions that contribute to their losses, and special features to help reduce losses in these power converters.

Overview of Power Delivery

Figure 4-5 illustrates an example power converter block diagram for a standard dual processor system. Block diagrams like this are commonly found in motherboard schematics.

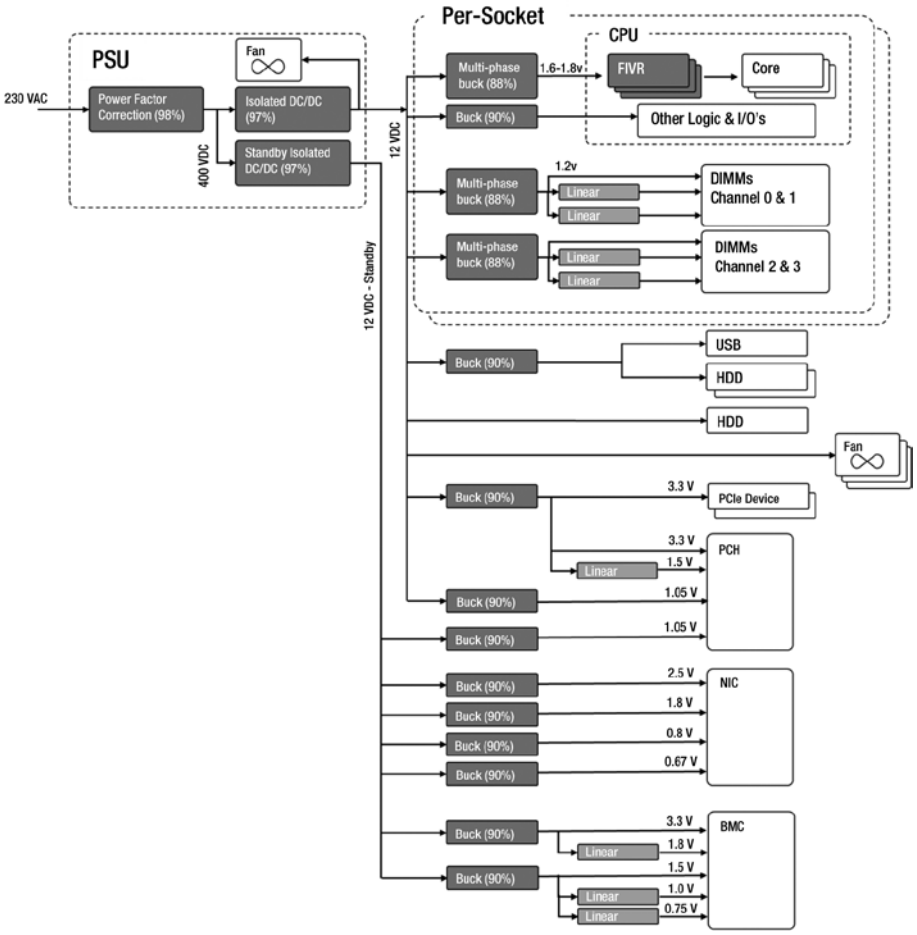


Figure 4-5. *Dual socket power conversion block diagram*

Power is first processed by a power supply and converted from AC to DC (see Table 4-21). The output DC power from the power supply is then converted to the various DC voltage levels required by different platform components (see Table 4-22). Power budgets must be determined for each component in the system so that sizing can be done at each stage of the power delivery network (see Table 4-23).

www.dbooks.org

Table 4-21. *Components in a Power Supply*

Component	Type	Typical Efficiency	Description
Power factor correction	Power converter in system AC/DC power supply	~98%	This is a power conversion stage inside the system power supply whose primary function is to provide power factor correction. This is the first stage converter. It provides a 400 VDC output voltage to the second stage in the power supply.
Isolated DC/DC stage	Power converter in the system AC/DC power supply	~97%	This is a power conversion stage inside the system power supply whose primary functions are to provide safety isolation for the AC input and provide a regulated DC output voltage that can be used by the system.

Table 4-22. *Types of DC/DC Power Converters*

Component	Type	Typical Efficiency	Description
Multi-phase buck	DC/DC switching power converter on the motherboard	80–90%	Power converter used to provide high currents at low voltages. Frequently converts 12 V to 1–2 V.
Buck regulator	Simple DC/DC switching power converter on the motherboard	~90%	Simple converters used to power lower power devices on the motherboard. Typical inputs of 12 V/5 V/3.3 V converting to outputs of 5 V to <1 V.
Linear regulator	Power converter used to power very low power devices on the motherboard	$\frac{Output\ Voltage}{Input\ Voltage}$	Very simple and low-cost converter that provides poor efficiency and therefore is used only for very low-power loads. Their efficiency is determined by the ratio of the output voltage to the input voltage.

Table 4-23. *Power Block Diagram—Loads*

Component	Description
Cores, uncore, DIMMs	These loads in the system are the primary power consumers and provide the core computing capabilities of the system.
LAN, PCH, USB, PCIe	These are lower power loads in the system that provide input and output to the system compute capabilities.
HDD	These are medium power loads that provide storage capabilities.
Active cooling	These are medium power loads that are primarily axial fans in the system. Other types of exotic cooling, such as liquid cooling, are also possible.

The block diagram contains switching power converters, linear regulators, and the loads. Almost all of the system power passes through three to four stages of power conversion to get from the 230 VAC input to the points of load. There are multiple reasons why the power passes through these series stages:

- The first step converts from AC to DC to provide power factor correction. Most digital circuits require DC power for operation.
- It is more efficient to transmit higher voltages over longer distances. This is why power is kept at higher voltages as long as possible.
- Low-power loads are powered by linear regulators. While these regulators are less efficient than more complex voltage regulators, they also have lower cost due to their simple design and small number of components. The loss in efficiency is small in the overall power consumption.
- The easily accessible portions of the platform must not expose technicians to dangerous sources of electricity (such as the AC input).

■ **Note** Transmitting power using higher voltages is more efficient. Wires used for transmitting power have resistance in them. Power is consumed because of this resistance and is proportional to the square of the current (Watts = I^2R). By increasing voltage at a fixed power, current is reduced, thus reducing these quadratic losses (Power = I * R).

www.dbooks.org

It's not uncommon to see a system with a total of 25–30 power converters in the system and on the motherboard. There are a few reasons for these many converters:

- Power efficiency can be improved by adding more regulators in order to reduce I^2R losses.
- Certain legacy functions and capabilities require specific voltage levels.
- System standard components (USB, HDD, PCIe adapters) require industry standard voltages that cannot be changed.
- Customized voltages are required for processors and other silicon devices in order to achieve high performance and power efficient designs.

Power Converter Basics

As discussed earlier, the system contains different types of power converters. Each of these has tradeoffs that can have a significant impact on the overall power efficiency of the platform.

First, energy can be transmitted either as AC (alternating current) or DC (direct current). Digital circuits require DC power to operate. AC power is commonly used to transmit power from power plants across the electrical grid because it is relatively easy (and inexpensive) to change AC voltages using transformers. AC power is used to distribute power within data centers as well, but it must be converted into DC power at some point in order to drive digital circuits.

AC/DC power conversion provides this mechanism. Different components in a platform require different voltages. DC/DC power converters change the voltage of DC power to match the requirements for each component. The main types of power converters used in standard server systems are boost converters, isolated buck converters, single and multiphase buck converters, and linear regulators.

System AC/DC Power Supply

The first components in the node power delivery network, the boost converter and the isolated buck converter, are integrated into the power supply. The basic schematic for these converters are shown in Figure 4-6. It is important to understand the basic functions of these converters to grasp the tradeoffs between efficiency and features.

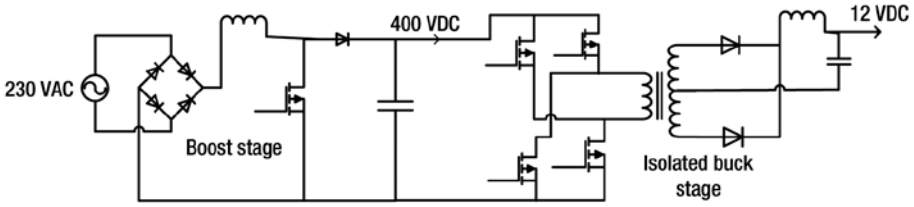


Figure 4-6. *Example AC/DC power supply schematic*

■ **Note** It is not a requirement that AC voltage be used as an input to a platform node and power supply. Other input voltages such as 380 VDC and 240 VDC are being used to help improve facility power distribution efficiency and availability. In these cases, the AC to DC conversion stage is not required and can be removed from the power supply to improve efficiency.

Both converters in the system AC/DC power supply are switching converters, which means MOSFETs are used to chop the input voltage into a square wave, and then they are filtered again to obtain a DC voltage. There is a PWM (pulse width modulation) controller that controls the duty cycle to maintain the required output voltage.

PSUs and the Boost Stage

The boost converter in the power supply maintains a regulated voltage to the isolated buck stage of the power supply. The boost's main purpose is to wave-shape the input current to provide power factor (PF) correction and lower current harmonic distortion (ITHD), resulting in improved power efficiency. Good power supplies achieve PF > 0.99 and ITHD < 5%. Since a boost converter requires that the output voltage always be greater than the input voltage, you typically see a boost output voltage of ~400 VDC ($>110\% * 240\,VAC * \sqrt{2} = 373\,Vpeak$).

■ **Note** Inputs other than 240 VAC are also possible. 277 VAC (one phase of a 480 VAC system) is becoming more common since it can be used in more efficient facility power delivery designs.

PSUs and the Isolated Buck Stage

The isolated buck stage of the PSU provides a few basic functions:

- A regulated output voltage (12 V in most server systems), which is used by down-stream DC to DC converters as well as fans
- *Galvanic isolation* (preventing current flow) between the AC input to the DC output as required by safety agencies
- *Ride-through capability*, which powers the system from its input bulk capacitor during short (½ to 1 cycle) loss of the AC input

The AC ride-through capability is important to keep in mind because this requires the isolated buck stage to maintain regulation on its output over a wider range of input as the bulk capacitor discharges. This tends to make the design of this stage less optimized for efficiency; however, it is required for reliability of the IT equipment.

www.dbooks.org

Redundant Power Supplies

Historically, many servers deployed redundant power supplies and redundant AC feeds to the system in order to improve reliability. Both of these are supported by using multiple power supplies in the system. A common design uses two power supplies. One PSU has enough power to power the system (sometimes at a lower performance), and a second power supply of the same wattage is used in parallel to provide redundancy in case either one fails. This is referred to as a *1+1 design*. The redundant power supplies normally share the load of the system. Since each power supply has its own AC input, this also provides 1+1 redundant AC feeds to the system. With more power supplies in the system a 2+2 or 3+1 redundant configuration is sometimes used. This can scale up to *N*+*N* or *N*+1 number of power supplies; where *N* power supplies are needed to power the system.

In recent years, certain classes of server deployments have focused on improved software resiliency. This is particularly true of large cloud deployments. In such situations, system failure is expected (at some low rate) and the software that executes on the system is robust to handle occasional failures. Conceptually, a problem that used to be solved with additional hardware (and procurement costs) is now being handled in software. Redundant power supplies are not necessary in such designs.

Shared Power Supplies

A single power supply (or even redundant supplies) can be shared by multiple nodes in some designs. A good example of such a design is with microservers. In such a design, the output power of the CPU is routed to multiple sets of voltage regulators associated with different nodes in the rack. In this case, the definition of a platform is somewhat blurred since the PSU is now a shared resource. One drawback of this approach is the *blast radius*, which refers to the number of nodes/components that are impacted if one fails.

PMBus

The power supply has become a key power measurement device in the IT equipment and is used by the facility to see how much power the IT load is consuming. The accuracy of these embedded sensors has improved to +/– 2% over a typical loading range of the system by using special metering IC in the power supply and by using calibration techniques on the manufacturing line. The power sensor in the power supply is used by Intel Node Manager (see Chapter 5) in conjunction with the processor RAPL feature (see Chapter 2) to control the system power. This allows the user to protect the facility infrastructure by guaranteeing that the system does not exceed a predefined limit.

DC to DC Power Converters

Once the power has been converted from a high VAC down to ~12 VDC, a number of additional DC to DC conversion steps are required so that each component in the platform is supplied with the voltage (and current) that they require. There are a number of different types of DC to DC converters that can be used in different situations.

Single-Phase Buck Converters

For medium power loads, single-phase buck converters are used to convert power from 12 V to lower voltages in the system. These may range from < 1 A to 30 A outputs.

Figure 4-7 provides an example schematic for a single-phase buck converter. The PWM controller converts the 12 VDC input to output voltage by switching the high-side MOSFET to chop the 12 V input. Then a DC voltage is reconstituted with the LC filter on the output. A low-side MOSFET is used to allow the inductor current to keep flowing through the output filter. The industry has optimized special components in these converters taking 12 V input to low-output voltages. The high-side MOSFETs are specially designed to handle the high switching voltages with very low duty cycles, and the low-side MOSFETs are optimized for the high duty cycles and high RMS currents demanded by their special requirements in the converters.

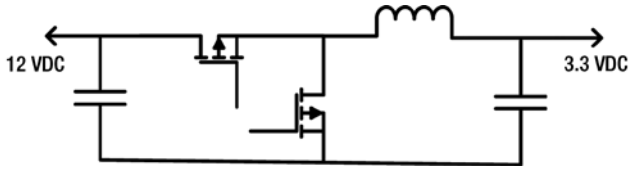


Figure 4-7. *Single-phase buck converter*

Motherboard Multiphase Buck Converters

In a standard server system, more than half of the system power goes to power the processors and memory. For mainstream motherboards, this power is supplied by multiphase buck converters to achieve high performance and small form factors. Requirements of these power converters drive their design to have very fast responses to load changes required by the processors and DIMMs, to maintain a tight voltage regulation on a low-voltage rail as silicon processes reduce their geometries, and to have a small footprint to fit on dense motherboards. All of these requirements challenge the efficiency of the motherboard VR designs. The use of multiphase buck converters has become the method for achieving the best design to meet all of these growing requirements and still maintain good efficiency.

Figure 4-8 shows a simplified schematic of the power stage for a multiphase buck converter. This example shows a four-phase buck converter. PWM controllers are available that have the flexibility to provide anywhere from two to six phases. Some can provide multiple output voltages. These multiphase converters have shared input and output capacitors. The PWM controller switches the phases similar to the single-phase buck; however, the controller switches one phase at a time. Therefore, for a four-phase buck converter, each phase is switched at 90 degrees from one another. This allows the controller to meet the high load transient and high current demands of the processors and DIMMs. The PWM controllers for these multiphase buck converters have added features to shed phases at lighter loads to save power (with no cost to software performance) and serial communication to communicate/manage these high power converters.

www.dbooks.org

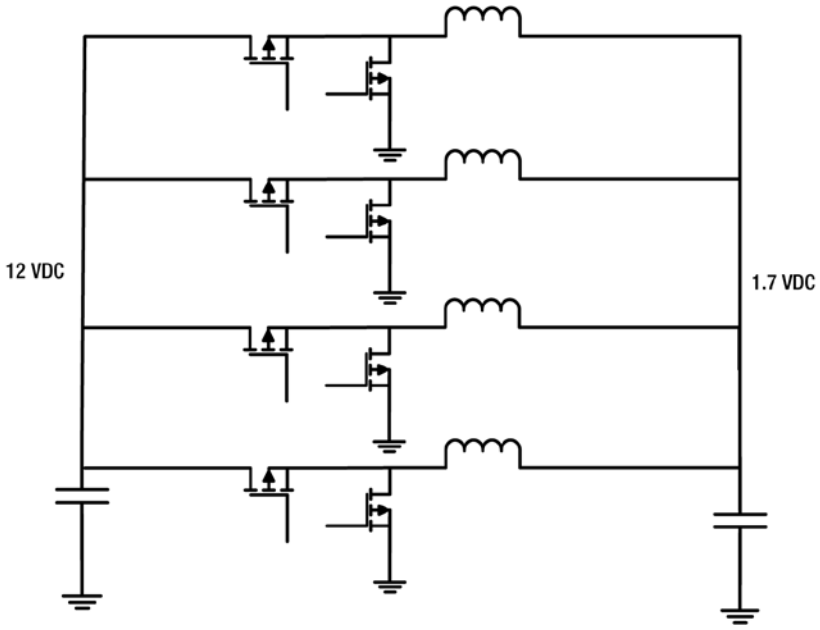


Figure 4-8. *Multiphase buck converter*

SVID

VRs used to power the main CPU voltage rails and DDR memory frequently support SVID (serial VID). SVID is a serial communication bus between the processor package and the voltage regulator controllers that is used for two main purposes to help improve the efficiency of the processor and manage power in the system:

- *Voltage set point:* The processor uses SVID to set the optimum voltage for the motherboard VR to power the processor. This can be used to set the static voltage for a given type of processor for certain rails. This can also be used to set the voltage to the cores in the processor dynamically as the P-state changes (see Chapter 2).
- *Power reporting:* The processor uses SVID to read the power from the VR on some systems. This way the processor can monitor how much power it and the memory is consuming, enabling RAPL (see Chapter 2).

One or more SVID busses can be used per CPU socket. They connect the CPU to all of the SVID-controller regulators that supply power to that CPU (or memory connected to that CPU). The SVID bus is a simple three-wire interface with a clock (the frequency can change across different platforms), an alert (interrupt), and a data wire. Multiple SVID busses may be required depending on the bandwidth demands of the bus for supporting both of the primary usage models just defined.

Motherboard Linear Regulators

Linear regulators convert a higher voltage to a lower voltage by dropping voltage across a series FET (field-effect transistor) operated in its linear range. The output is controlled by the FET's gate voltage. Linear regulators are used sparingly since they are lower efficiency. Their efficiency is determined by the ratio of output voltage to input voltage plus a small quiescent power. Linear regulators can be a good design choice for very low power supplies where losses are not significant in the big picture or when a very small voltage drop is required.

Integrated Voltage Regulators

The Intel Haswell processor integrated the last voltage regulator stages into the processor package with a new capability called Integrated Voltage Regulator (IVR). This added power conversion stage brings with it advantages that outweigh the disadvantage of adding another series power conversion stage.

- *Max current reduction.* Designing motherboard voltage regulators with high maximum currents can be cost prohibitive. By providing the die with a higher input voltage (and by using IVR to step the voltage down for use by the circuits), the max current provided to the die decreases.
- *Higher input voltage to the processor resulting in smaller power delivery losses in the platform.* IVR allows a higher voltage to be delivered to the processor package while still maintaining the required lower voltages at the chips since the IVR power converter controls the chip power. In Haswell, the package input voltage is maintained at about 1.8 V, about twice the voltage needed by the circuits in the package. By running at twice the voltage, the current required to provide a given level of power is cut in half. Lower current results in less voltage loss between the VR and the package (in the platform), improving the overall platform power efficiency.
- *Tighter voltage regulation resulting in lower voltage guardbands and lower power operation.* IVR brings with it tighter voltage regulation at the chips since it is physically closer to the chips. This means the voltage may be kept lower at the chips since less margin for parasitic inductive drops needs to be allowed for—the lower the voltage, the lower the power consumption of the silicon (lower leakage and lower active power).
- *IVR provides cost-effective voltage control of small subcomponents within a die such as an individual core.* This enables features like per-core P-states (see Chapter 2). It also enables the voltage levels to be optimized for each of these subcomponents.

www.dbooks.org

One drawback of IVR is that the power losses that existed in motherboard VRs are moved into the IVR on the CPU die. Although this may result in a net power win for the platform, it does increase the power on the CPU die, which can lead to challenges with thermal density and cooling. When you compare similar SKUs on Haswell E5 (with IVR) and Ivy Bridge E5 (without IVR), you will notice that the TDP power has increased on Haswell. These changes were primarily driven by the increases in CPU power from the IVR integration.

Power Management Integrated Circuit

The term *Power Management Integrated Circuit (PMIC)* is applied to integrated circuits that have multiple power conversion controllers in one small package; they also may contain integrated switches for supporting switching buck converters. The integration of many converters into one package helps to reduce the size and has traditionally been used in the small form factors of mobile devices like phones and tablets. PMICs are now being applied in server computers to help keep the size small while still supporting the many lower power rails on the motherboard. These PMIC may be used to power LAN, PCH, and BMC devices on a standard server board or the memory and processor rails on a microserver with a SoC package. The reason to use PMIC is not to reduce losses in the system, but to reduce the size of the power converters.

Power Conversion Losses

Now that we have reviewed the various types of power converters and their applications, this section will take a holistic look at power converters to understand what causes losses. It will also explore system level design tradeoffs.

Some energy is always lost in transmission through wires because of the resistance in those wires. There are losses due to the currents passing through resistances; these are the condition losses and are proportional to the resistance and the square of the current (power = current^2 × resistance). If we consider only these resistive losses, we would expect the losses at very light loads (like those found when a system is at idle) to drop to very low power; however, this is not the case. We need to consider two other types of losses that occur in switching power converters; these have been commonly referred to as *proportional losses* and *fixed losses*.

Proportional losses are losses in the power converter that increase linearly with the output power of the converter (or output current, since power converters are voltage regulators). These are elements like diodes that have a loss equal to the forward drop of the diode times the current through the diode. The proportional losses of power converters are not very interesting since this is not a dominant element at any load.

The *fixed losses* of the power converter are very significant at light loads. Fixed losses in a power converter are caused by elements such as the switching of the MOSFET parasitic capacitances, the switching of currents through the power components, and the transformer core hysteresis losses in the power supply. These fixed losses are always present whenever the power converter is working. Table 4-24 provides an overview of the types of power delivery losses on a platform.

Table 4-24. *Types of Power Conversion Losses*

Type	Description
Conduction losses	Power losses that are caused by current passing through resistive elements.
Proportional losses	Power losses that are caused mainly by the forward drop of diodes. These are the least significant of the three types of losses.
Fixed losses	Power losses that do not change with the output load on the converter.

Motherboard VRs

Each of the different types of motherboard VRs exhibits different power efficiency profiles. The behavior and efficiency of different types of voltage regulators changes as the load (current demand) of the regulator changes. This section will evaluate different types of VRs by looking at their power losses both in terms of power (watts) as well as their efficiency.

■ **Note** Voltage regulator efficiencies typically appear poor at low utilizations. However, it is important to note that the actual power losses in these conditions are relatively small in absolute terms compared to the losses at higher utilizations.

Single-Phase Buck Converter

Figures 4-9 and 4-10 show the loss curve and efficiency curve for a single-phase buck converter that is capable of 30 A maximum output load and converts 12 V input to 1.7 V output. A second order polynomial trend line of the losses versus the output current closely fits the loss curve. These three coefficients represent the squared ($0.052x^2$), proportional ($0.045x$), and fixed ($+1$) losses of the single-phase buck converter where x is the output current of the VR. The plot also shows these three loss elements separately to see how they contribute across the output load. The fixed losses dominate at lighter loads less than 10 A and the squared losses dominate at heavier loads of greater than 20 A. The effects of the fixed losses cause the efficiency to drop very quickly as load decreases below 10 A, and the efficiency tails off at heavier loads due to the effects of the squared losses.

www.dbooks.org

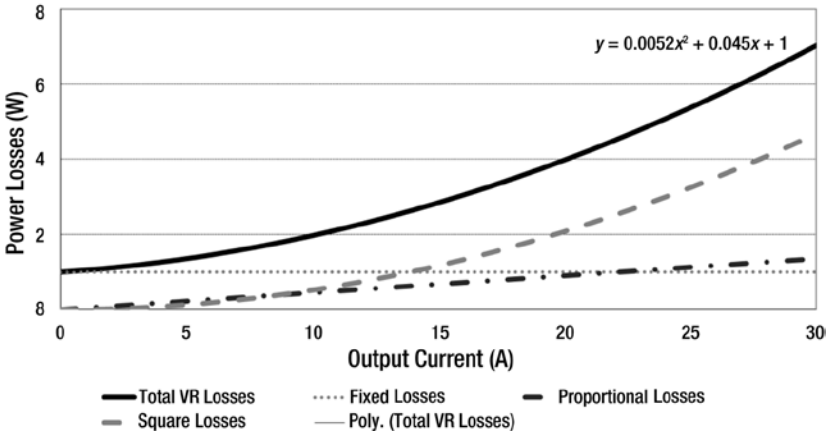


Figure 4-9. *Example single-phase buck converter losses*

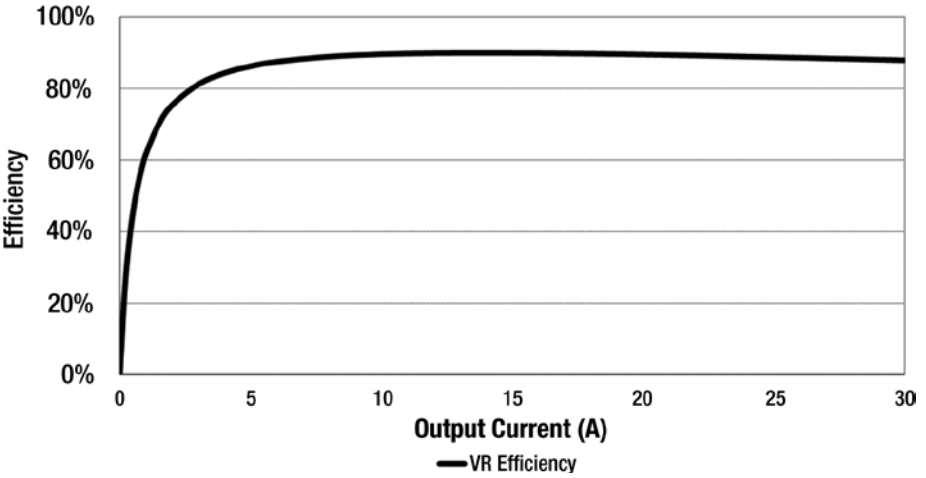


Figure 4-10. *Example single-phase buck converter efficiency*

■ **Note** VRs must be sized for the worst-case possible current demands of the loads they power in order to avoid system failure. However, typical steady-state current demands, even under heavy load, are common at much lower utilization levels. As a result, the efficiency tail-off that is observed at higher utilizations is generally less significant to the overall power delivery efficiency than the efficiency losses at low utilizations.

Multiphase VR Losses

Multiphase VRs are typically used for heavier loads on the motherboard, like memory DIMMs and processor cores. If we consider the same 12 V to 1.7 V buck converter, but expand it to three phases to support up to 90 A, we see a different loss curve compared to the single-phase converter. Figure 4-11 shows the losses in the three-phase converter are higher than the single-phase converter. This is due to the additional fixed losses from the additional phases and added switching losses of the extra phases.

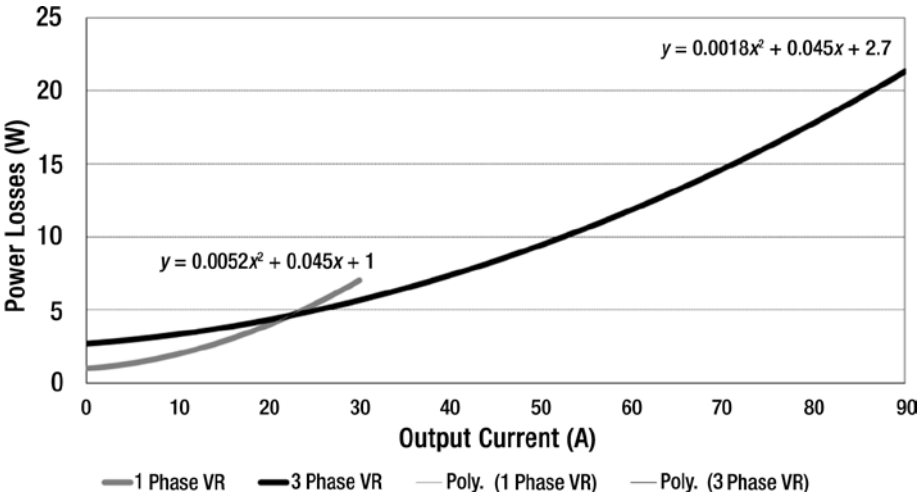


Figure 4-11. *Example multiphase VR power losses*

At loads greater than 20 A the three-phase converter starts to have lower losses. The squared component (from conduction losses) is smaller with the multiphase VR (0.0018 vs. 0.0052). This is because the current now has about a third of the resistance to pass through.

Comparing the efficiency curves shows how the single-phase converter is better at lighter loads and the three-phase converter is better at heavier loads. Note that this example is provided without phase shedding (discussed momentarily), which can improve the efficiency of multiphase VRs at low utilizations.

Phase Shedding

Figure 4-12 illustrates that a single-phase VR can provide better efficiency at low current demands when compared to a multiphase VR that is capable of much higher max current. *Phase shedding* is a feature on some multiphase VRs that is intended to provide the best of both worlds: good, light load efficiency of a single-phase buck converter and the power/efficiency advantages of the multiphase buck converter at heavier loads. In present systems, the phase shedding has mostly been controlled by the processors; the phases are shed when the processors know their power requirements are less than a single-phase

www.dbooks.org

capability. In newer PWM controllers, the controller automatically turns off the phases as it senses the load dropping and turns on phases as loads increase. This is referred to as *auto-phase shedding*. Auto-phase shedding can be far more efficient than CPU-managed phase shedding, because the CPU is not always aware of the immediate current demands and must request phases assuming some worst-case condition.

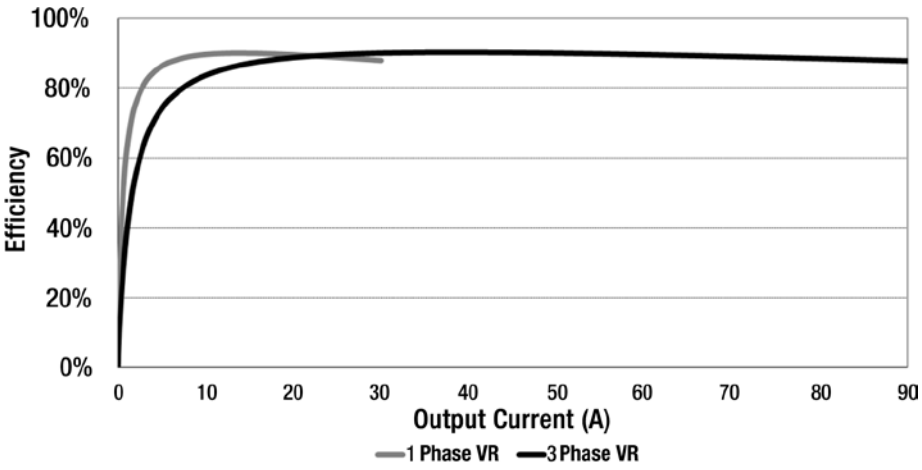


Figure 4-12. *Example multiphase VR efficiency*

Diode Emulation and Burst Mode

Two other methods used to help reduce VR losses at very light loads are *diode emulation mode* and *burst mode*. Diode emulation mode turns off the low-side FET switch and instead uses the body diode in the FET, saving the switching losses in the low-side FET. This is used only at very light current demands. Burst mode reduces losses by skipping switching cycles to effectively reduce the switching frequency of the converter, helping to further reduce switching losses, again at very light current demands.

■ **Note** In typical servers, the current demand does not drop low enough to take advantage of diode emulation, even when the system is completely idle. As a result, this feature is not as commonly supported. Phase shedding provides the bulk of the efficiency improvements at low utilizations in server VR designs.

System Power Supplies (AC/DC)

System power supplies have similar power losses and also can be accurately modeled as a second order polynomial made up of squared losses, proportional losses, and fixed losses. In most server systems, manufacturers offer multiple power supply wattage ratings that can be used in the same system. This allows the users to optimize the cost of the power supply for the configuration they plan to use in the system (e.g., processor performance, memory size, storage size). The selection of the power supply wattage also affects the power consumption of the system. Using a properly sized power supply in the system can help reduce the system power. The use of redundant power supplies in the system to improve availability also affects the efficiency of the system.

Figure 4-13 shows the losses in a 750 W power supply with the fixed, proportional, and squared losses broken out. This is an 80-Plus platinum-level-efficient power supply.[7] As with the motherboard VR, at low loads (less than ~30% of peak), the fixed losses dominate and cause the efficiency to drop off. At moderate to high loads (greater than ~50% of peak), the squared losses dominate and cause the efficiency to drop off.

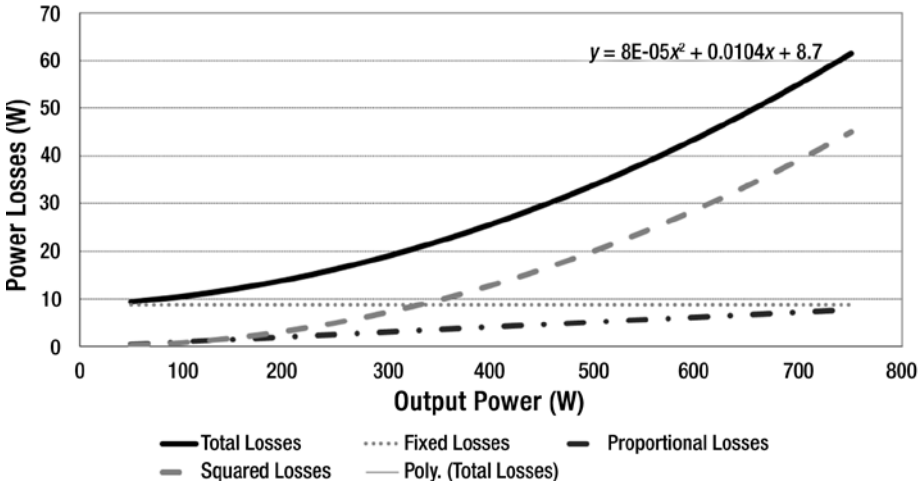


Figure 4-13. *Example 750 W PSU losses (230 VAC)*

■ **Note** PSU losses increase with lower input voltages. The charts in Figure 4-14 are measured with a 230 VAC input (high line). At lower input voltage, such as 120 VAC, the efficiency is reduced by about 2%. This is due to the higher currents in the power factor correction stage of the power supply.

---

[7] >94% efficiency at 50% load based on requirements documented at www.80plus.com.

www.dbooks.org

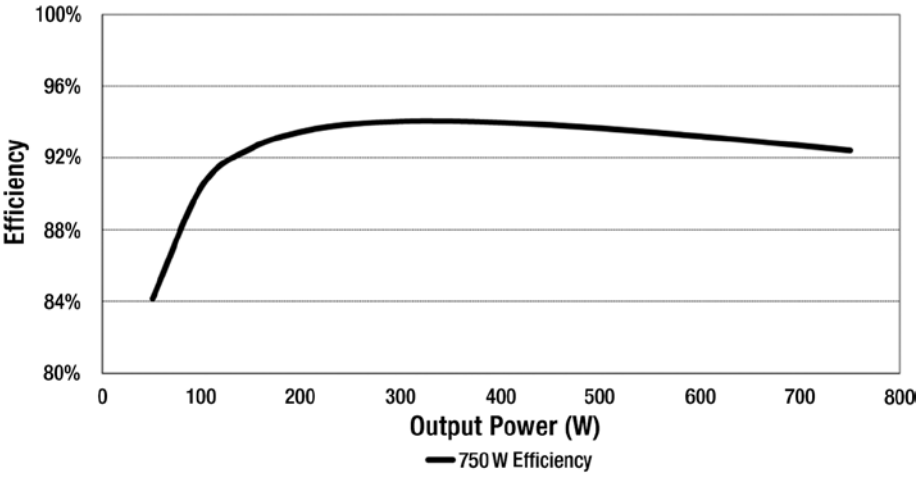


Figure 4-14. *Example 750 W PSU efficiency (230 VAC)*

Right-Sizing Power Supplies

Next we will consider the tradeoffs in using power supplies with higher and lower power ratings. Figure 4-15 shows the loss curves for four different power supplies: 460 W, 750 W, 1200 W, and 1600 W. These are all platinum efficient per 80 Plus.[8] The squared, proportional, and fixed loss coefficients are also shown as a comparison. Two notable conclusions can be drawn from this chart:

- Power supplies with lower wattage ratings have lower fixed losses.
- Power supplies with higher wattage have lower squared losses.

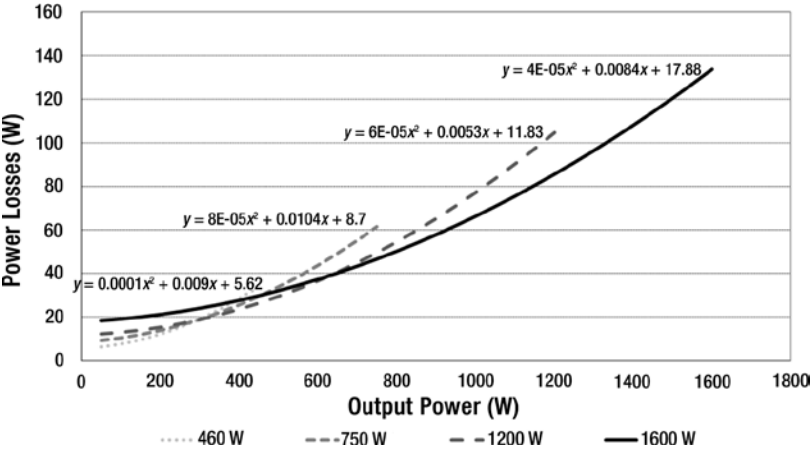


Figure 4-15. *Example PSU losses for different power ratings*

[8] 80 Plus is a voluntary certification program for PSUs.

■ **Note** Power supply selection can have an impact on the power consumption of a system. Systems that run at low utilizations will experience the best power efficiency using power supplies that are just large enough for the system. On the other hand, power can be saved by selecting an oversized power supply if system utilization tends to be heavy.

The following two examples show how differently sized power supplies can provide benefits to power efficiency depending on the typical load of the system. Larger power supplies can be more power efficient in systems that, on average, run at higher utilizations.

> *Example 1:* A system load of 100 W on the output of the power supply produces 7.7 W losses in the 460 W power supply, whereas the 1200 W power supply produces 13.0 W losses, a 5.3 W savings using the smaller power supply.
>
> *Example 2:* A system load of 700 W on the output of the power supply produces 55.2 W losses in the 750 W power supply whereas the 1600 W power supply produces 43.4 W losses, a 8.2 W savings with the larger power supply.

Closed Loop System Throttling (CLST)

When right-sizing your power supply to the system configuration and the workloads you plan to run, you must consider the system reliability. If some abnormal condition occurs on the system (like running a higher power workload) the system cannot shut down due to an overload on the system power supply. Many systems running Intel Node Manager and a PMBus power supply have a protection feature called Closed Loop System Throttling (CLST). This feature will throttle the system power/performance if the power supply senses an overload warning condition. This quick reduction in load will protect the power supply from shutting down. Therefore, CLST provides protection against unexpectedly higher system power consumption. This gives you the protections needed to maintain good system reliability while using a lower power supply rating. This throttling is very aggressive and can result in significant performance loss. As a result, it is important that you budget sufficient headroom in the power supply selection to compensate for increases in power demand that may occur over the life of the server (software updates resulting in higher power draw, increased temperatures, etc).

Losses in Redundant Power Supplies

When considering redundant power supplies, remember that the system power supplies will share the system load, which will change the overall power supply losses in the system. Figure 4-16 shows an example of the power supply losses for a 750 W in a non-redundant single PSU configuration along with a redundant 1+1 PSU configuration. The x-axis in the plot is the total load on all power supplies in the

www.dbooks.org

system. You can again see that for heavier loaded systems (> ~500 W, in this example), the redundant 1+1 system with two power supplies in the system have lower losses in the power supplies. And for lighter loads (< ~500 W) the redundant power supply system has higher losses in the power supplies than the system powered by a single power supply.

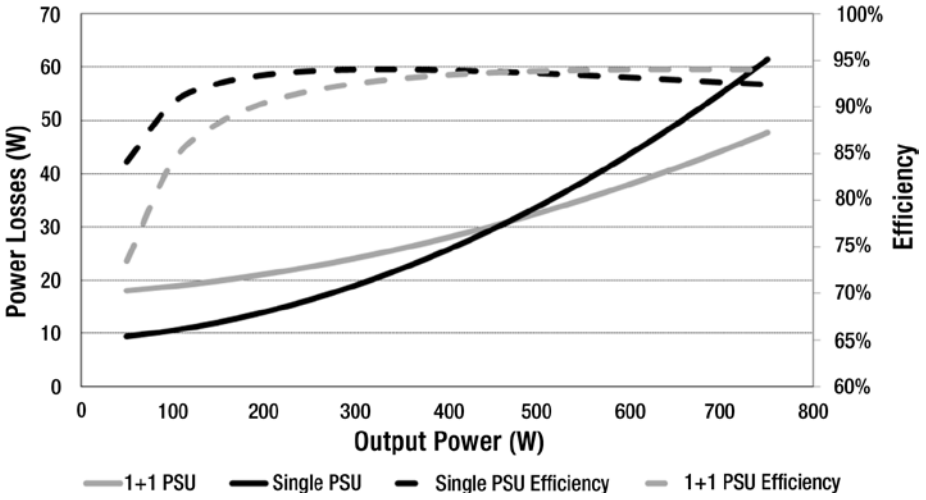


Figure 4-16. *Example PSU losses with redundant power supplies*

■ **Note** System availability and reliability is generally a high priority for server customers. The efficiency loss associated with redundant power supplies is generally an acceptable cost, and many systems make use of redundant power supplies as a result.

The relative efficiency of redundant power supplies versus single power supplies is dependent on the specifics of the power supplies in question, as shown in the following two examples: one shows a more efficient single power supply, and the other lists a more efficient redundant power supply.

> *Example 1:* At a load of 200 W on the system, the single 750 W losses are 14.0 W and the 1+1 power supply total losses are 21.1 W, a 7.1 W savings in losses for the system with a single power supply.
>
> *Example 2:* At a load of 650 W on the system, the single 750 W losses are 49.3 W and the 1+1 power supply total losses are 41.1 W, a 8.2 W savings in the losses for the system with the 1+1 configuration.

Power Supply Cold Redundancy

You can see by the preceding power supply redundancy loss examples that to achieve the best power efficiency in all cases, it would be best to have something similar to what was discussed for the motherboard VR phase-shedding feature. So, at lighter loads, the system runs from one power supply (but still maintains redundancy), and at heavier loads, the system runs from both power supplies in a load-sharing mode. This can be achieved by a feature supported by many server systems today, which is sometimes referred to as *cold redundancy*. In this case, one power supply is powered off into a cold standby state automatically at lighter loads. The cold standby state still allows the power supply to power on quickly if the active power supply fails. This maintains the system power supply redundancy feature. Then, at heavier loads, the cold standby power supply powers on automatically to share the load and maintain the lowest possible losses in the power supplies.

Thermal Management

Typical servers in data centers consume a large amount of electricity, turning it into heat. Extracting this heat from the data center consumes a non-trivial amount of overall data center electricity. Over time, the efficiency of cooling has improved significantly, reducing the overall contribution to power. However, it is still a major factor in energy consumption.

A server cooling system must ensure that each and every component meets its specification. Most components have damage, functional, and reliability temperature specifications as seen in Figure 4-17. A well-designed thermal management scheme must ensure compliance to the specifications while also not over-cooling and wasting power. In most cases it is impractical to design a system to handle every possible workload under all possible combinations of extreme conditions, including fan failures, high-room ambient temperatures, and altitude.

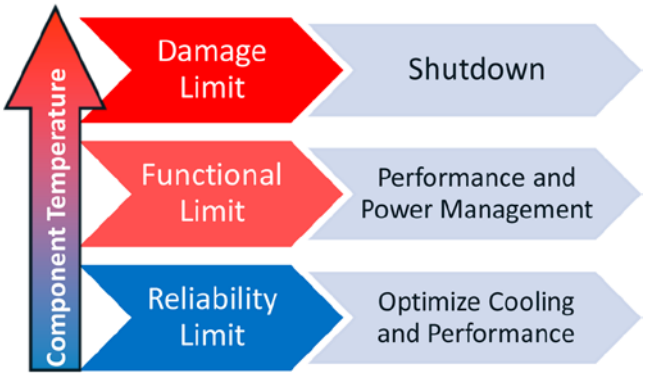


Figure 4-17. *Component temperature specifications and thermal management*

www.dbooks.org

The functional limit is normally aligned with maximum system utilization whereas the server is exposed to a worst-case corner of the allowable range of the environmental class (temperature, altitude, humidity) for which the server has been designed. At lower utilizations, the system is maintained at a lower temperature in order to reduce component wearout that can occur if higher temperatures (at or near the functional limit) are sustained for long periods of time.

A well-designed server will have thermal management to ensure compliance to those specifications either directly through the cooling design implementation, or in combination with the thermal management system. Component temperature is driven by three factors in an air-cooled system defined in Table 4-25: system ambient, air heating, and self heating. These are illustrated in Figure 4-18. Table 4-26 provides some common terms used for heat transfer.

Table 4-25. *Types of Heating*

Type	Description
System ambient[9]	*Inlet temperature of the system* This includes any rack effects, which can increase the temperature delivered to the platform node.
Air heating	*Increase in air temperature due to upstream heat sources in the platform* This is affected by component placement, upstream component power dissipation, air movers, and local air delivery.
Self heating	*Increase in component temperature above local ambient due to the heat dissipated on the device of interest* This is driven by component packaging, power dissipation, and thermal solution (e.g., heat sink).

[9]Defined in the ASHRAE (American Society of Heating, Refrigeration, and Air-conditioning Engineers) Thermal Guidelines for Data Processing Environments.

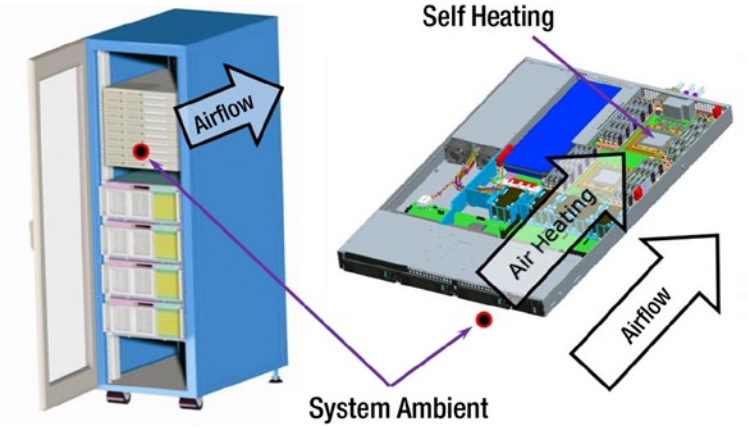


Figure 4-18. *Local ambient, air heating, and self-heating of second socket*

Table 4-26. *Common Heat Transfer Terms*

Type	Description
Conduction	The heat transfer at the molecular level between adjacent particles. Heat from a die is conducted through the surrounding packaging until it is delivered to a heat transfer surface (like a heat sink).
Convection	The heat transfer through random molecular motion and bulk movement of a fluid (or air). Airflow in a server is an example of convection.
Radiation	The heat transfer through electromagnetic waves; generally negligible in server heat transfer due to the dominance of forced convection.

Most server processor dies are connected to a substrate made of FR4 (a glass-reinforced epoxy laminate) enabling simple integration using a socket that enables removal and replacement of the processor. To facilitate heat sink attachment, an integrated heat spreader is attached on top of the package. Component heat is transferred by conduction to the heat spreader and removed by forced convection.

When a heat sink is used on a component, a thermal interface material (TIM) is required to fill the air gaps between the component and the heat sink. The TIM has much higher thermal conductivity than air.

Figure 4-19 shows the typical packaging of a processor with an integrated heat sink (IHS). The IHS serves to protect the die, spread the heat, and provide a mounting surface for a heat sink. Heat is primarily conducted through the first TIM (TIM 1) to the IHS and out through the second TIM (TIM 2) to the heat sink. Thermal paste between the CPU package and the heat sink is an example of a TIM 2. Most servers use forced convection created by a fan to provide higher local velocities, thereby enhancing the convective heat transfer out of the heat sink.

www.dbooks.org

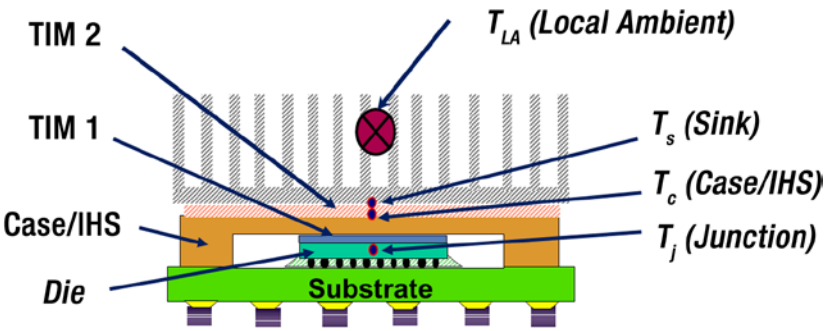


Figure 4-19. *CPU packaging thermal terminology*

When designing an air-cooled system, the thermal engineer must consider a number of factors contributing to the component temperature. Through a careful understanding of the critical components, their specifications, and placement requirements, the thermal engineer can optimize layouts to maintain the lowest cost, highest efficiency, and highest performance solution. So-called shadowing of components results in significantly increased cooling difficulty and the lowest cooling capability. Shadowing implies that the air heating in the following component temperature equation will be relatively high, resulting in costly thermal solutions and high fan power.

Components with high power density (power/area) require thermal enhancement, such as a heat sink or heat spreader. Either of these devices spreads the heat to a larger surface area enabling significantly improved convective heat transfer.

The following equation describes how power, air heating, and ambient temperatures impact the temperature that is exposed to the package. The actual silicon die (and transistors) are exposed to even higher temperatures than the T-case.

$T_C = \Psi_{CA}$ x Power + System Pre-heating + External Ambient

where

- T_C (T-case) is the case temperature of the component.
- Ψ_{CA} (psi-CA) is the thermal characteristic of the heat sink as measured from case (C) to ambient (A) and in units of °C/W. The lower the Ψ_{CA}, the better the thermal performance of the cooling solution, since the component (case) temperature will be closer to the ambient temperature at a given power consumption.
- Power is the power dissipated (consumed) by the component.

Component and heat sink convective thermal performance is proportional to the inverse of airflow, as shown in the example characteristics shown in Figure 4-20. This means that the cooling efficiency (Ψ_{CA}) improves significantly with airflow up until a point (somewhere between 10 and 15 CFM in the illustration). After that point, significant increases in airflow (at high power cost) will only provide small benefits to the actual cooling. Fan power is proportional to the cube of airflow (and fan speed). Operating in the conduction-dominated region of a heat sink can significantly increase the power consumption (and inefficiency) of a system.

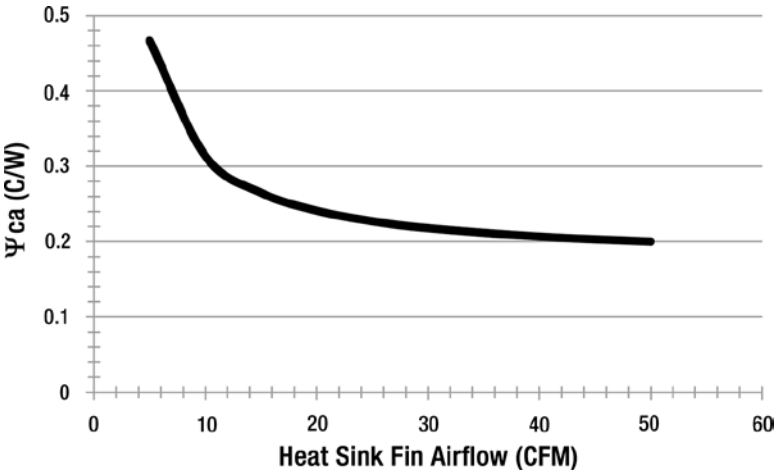


Figure 4-20. *Example heat sink performance*

■ **Note** Cooling efficiency is non-linear with airflow. Significant increases in fan speed (and fan power) may only yield slight improvements in cooling once a heat sink has reached its maximum capabilities.

System Considerations

The platform design team must carefully consider the components, configurations, usage models, environmental conditions, and the system-, rack-, and room-level airflow protocols to achieve an optimal cooling solution. These design considerations must be evaluated against the cost, performance, and energy objectives of the solution.

■ **Note** Running a system at higher temperatures will increase the leakage power of the CPU (and other devices in the platform). However, the power savings from running with reduced cooling typically far exceed the increases in device leakage power.

Component selection and placement detail will drive the design and consequently are the most critical elements to consider during the design phase. One example is the selection of memory technology to be supported. An entry-level server designed to support the highest capacity and frequency memory could burden the system design with expensive fans that are never needed by most customers. All components must be similarly considered including the power range under which the components must function.

www.dbooks.org

■ **Note** Increased CPU leakage power from higher temperatures can reduce turbo performance. However, the performance returns from over-cooling a platform are generally not large and can be cost prohibitive. Reducing the temperature by 20°C may only increase performance by a few percent (if at all).

In many platform designs, component placement is primarily driven by electrical routing considerations. Lengths between key components must be minimized to ensure signal integrity and meet timing requirements. Placement for thermal considerations matters but is not the foremost driver during the board layout process. The thermal engineer must provide the guidance to the board design team to enable solutions that have a reasonable chance for success while not necessarily being thermally optimal. Examples of systems that vary in cooling difficulty are shown in Figure 4-21, where the system on the left has thermally shadowed memory and processors whereas the system on the right does not. *Thermally shadowed* refers to a component being downstream from another component in the airflow. In such a design, the shadowed components are exposed to higher temperatures.

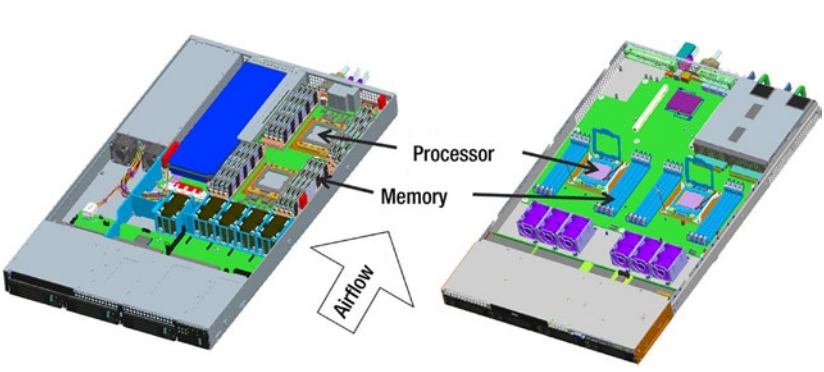


Figure 4-21. *Example board layouts*

Thermal shadowing is commonly used in dense multi-socket platform designs. Because the first processor heats the air before it gets to the second processor, the ambient temperature of the second processor is higher. The cooling solution must compensate for this increase in ambient temperature. This frequently results in more expensive heat sinks and higher fan speeds, which increases both procurement costs and power consumption. The thermal requirements of higher densities come at a power/performance efficiency cost.

■ **Note** Components that are in the thermal shadow of other high-power components frequently operate at higher temperatures and therefore consume more leakage power. With CPUs, this increase in power can result in different levels of turbo being achieved if equal power is allocated across the two sockets.

The design engineer must thoroughly understand the expected airflow paths and optimize the airflow delivery accordingly to maximize energy efficiency of the thermal subsystem. Selection and usage of the air moving devices must be matched and designed to the server design. Tradeoffs between air movers and heat sink design must be performed to find the optimal design points for both. The cooling performance, power consumption, acoustic signature, fan reliability, and redundancy features are important characteristics that factor into the overall solution.

Component Thermal Management Features

Power management features are used to perform power-performance limiting that enables a component to stay within temperature limits. Sensors create the data necessary to trigger power management. Processors, memory, and some chipset components contain sophisticated thermal management capability and are discussed in the following sections.

Processors

Processors have three high-level temperature points as shown in Table 4-27. These temperature values vary across different products, and the values shown provide an example of typical values.

www.dbooks.org

Table 4-27. *Notable Processor Temperature Levels*

Level	Description	Typical Temperature
TCONTROL	Above this temperature, fans should be running at full speed in order to ensure the long-term reliability of the processor. Between TCONTROL and PROCHOT, the fans will all be running at full speed. These conditions typically occur when the processor is running at full utilization and ambient temperature is high.	~5°C–10°C below prochot
PROCHOT (DTSMAX)	Max temperature at which the processor functionality is guaranteed. Autonomous thermal management algorithms inside the processor (see Chapter 2) will work to ensure that this level is not exceeded. Between PROCHOT and THERMTRIP, processor functionality is not guaranteed. Data corruption (silent or detected) or system hang may occur.	~80°C–100°C
THERMTRIP	Catastrophic shutdown temperature. Above this temperature, irreversible physical damage may occur to the processor. This is protected by a combination of the processor and the platform.	~125°C

Memory

Similar to processors, power management features are used to manage potential excursions above unsupportable temperature limits on DIMMs. Because the memory controller is now contained in the processor, the processor determines the memory's thermal state and activates power management features. Thermal sensors on the DIMMs are accessed by the processor, and memory traffic regulation can be activated as needed.

The data retention time of DRAM devices used on DIMMs is temperature dependent. Increasing the memory refresh rate allows operation at higher temperatures. Operation at that higher temperature is called extended temperature range (ETR) and is supported by most volume DRAM manufacturers. By enabling higher temperature operation, one can reduce cooling costs of the platform. This does come at a small cost of DIMM power from the extra refreshes and some small performance loss, but the resulting fan power savings implies higher power efficiency at the platform level. As memory temperatures increase beyond the ETR threshold, memory thermal throttling features in the CPU will engage. See Chapter 3 for more details.

Platform Thermal Management

Thermal control enables optimization of system performance as a function of usage, configuration, cooling capability, and environment. Underlying this optimization is the parallel use of fan speed control and power management to meet the customer's requirements. Some customers may desire maximum performance and may not want to be concerned with cooling costs, while others may be willing to trade off a small amount of performance under certain circumstances in order to achieve better power efficiency (and lower cooling costs).

Components and their specifications are the primary drivers in a server's thermal design (e.g., heat sink, fan selection, and airflow management). The thermal engineer can create a superior thermal design, but without a thermal management system to provide real-time optimization, that design may be acoustically unacceptable or highly inefficient. True superiority quite often lies in the thermal management scheme and its capability for delivering precisely the performance needed and meeting the component specifications while consuming the lowest amount of power.

Platform thermal management enables optimization in four areas:

- Operation within component thermal limits
- Maximization of performance
- Minimization of acoustic output
- Minimization of wall power consumption

All server components are designed to handle thousands of thermal cycles due to the natural temperature variation that occurs as a result of workload demands. Servers can go from idle to high usage many times a day and must be capable of years of operation under this type of variation, resulting in wide temperature extremes on the components.

The thermal management (TM) system manages component temperatures, performance, power consumption, and acoustics using two primary mechanisms:

- Fan speed control (cooling delivery)
- Component power-limiting features (e.g., P-states and memory throttling)

With some servers the initial setup during boot time enables the end user to configure the system to preferentially favor acoustics, power efficiency, or performance.

www.dbooks.org

Thermal Control Inputs—Sensors

Sensors provide the inputs to the control scheme. Table 4-28 provides an overview of some types of sensors used for platform thermal management.

Table 4-28. *Types of Sensors Used for Thermal Management*

Sensor Type	Description
Direct temperature	On-component sensor(s) found on processors, memory, hard disk drives (HDDs), chipset, GPGPU, etc.
Indirect temperature	Off-component, discrete sensor used to directly measure air or board temperature. This information can be correlated to components without sensors.
Power or activity	Power can be used to estimate the temperature of different components of a platform (in conjunction with platform characterization and other temperature sensors). It can also be used by algorithms to optimize the overall platform power.
Fan speed	Used for ensuring that a fan is operating within design parameters.
Fan presence	Used for detecting whether a fan is populated (e.g., redundant configuration).

Different components in the platform use different types of sensors for monitoring temperature (see Table 4-29).

Table 4-29. *Platform Component Sensor Types*

Component	Sensor Description
Processor	A server processor has many thermal sensors but only exposes the max temperature to the platform thermal management. Multiple sensors are strategically placed to enable the processor's own power management features to engage as necessary to ensure that silicon temperature does not exceed the point to which the processor was qualified and tested, but also to eliminate inaccuracy in determining actual die hot spot temperature.
Memory	Most server DIMMs have an on-PCB (printed circuit board), discrete thermal sensor. Thermal sensor temperature is highly *correlated* with DRAM temperature, thereby enabling a single sensor to cover all components on the DIMM. Some DIMMs have a buffer, which may also have a separately accessible thermal sensor.
Chipset (and other silicon devices)	Many silicon devices have an accessible sensor for use in TM. Some limited thermal management may be available locally on these devices, but they are used primarily for fan speed control and catastrophic protection.
Hard drives	Hard drives contain thermal sensors that are accessible through a drive or RAID (redundant array of inexpensive drives) controller.
Voltage regulators	Nearly all high-powered voltage regulators have a local thermal sensor to protect the components in the VR region. Historically this has been primarily for high-temperature protection.
PCIe cards	In some cases the PCIe card supports sensor capability, which is available to the server for thermal management. However, this is not common and, as a result, cooling must be sized to handle any possible card that can be installed. Indirect sensors are sometimes used to infer PCIe temperatures.
Power supplies	Most power supplies have their own cooling (internal fans) and manage their cooling without system intervention.

www.dbooks.org

Figure 4-22 provides an example of how thermal sensors are distributed across a platform.

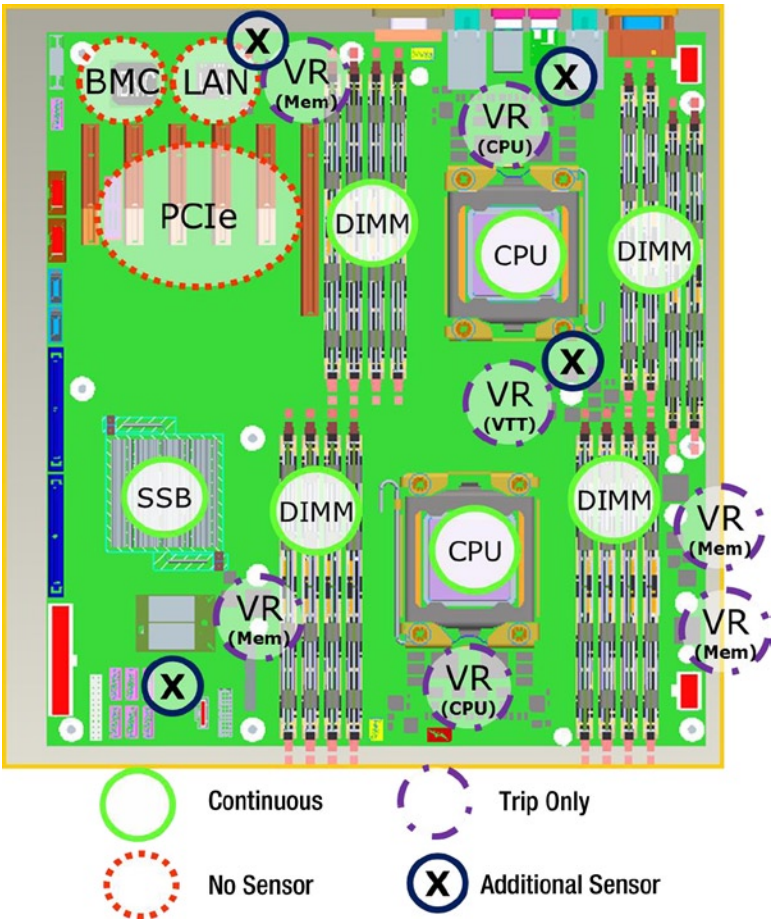


Figure 4-22. *Example platform thermal sensor layout*

Voltage Regulators

Voltage regulators (VRs) can be made of multiple discrete components on a board, and a thermal sensor is generally placed near the component that is expected to exhibit the highest temperature. In the past, the sensor primarily provided functional protection with little available usage for fan speed control. More fine-grained TM implementations have the capability for using the VR sensors in fan speed control algorithms. VRs can support

- OTP (over-temperature protection), which results in an immediate shutdown
- Prochot, which is a connection to platform Prochot to cause the CPU to engage in heavy throttling when the VR is hot
- VRHOT, which is an alert on SVID to tell the CPU to throttle

Power Supplies

Power supplies have their own thermal sensors and fans that are used for autonomous thermal management. They have temperature protection mechanisms that can shut down the system when a catastrophic condition is detected. The power supply's fans can supplement the server's cooling in certain conditions. As a result, the platform TM system can sometimes override the power supply fan control in order to drive higher fan speed as necessary to cool system components.

Fan Speed Control and Design

Optimizing the speed of fans in a system can result in significant improvements in power efficiency. Simply running the fans at max speed is an easy way to ensure that the system operates within its specifications and provides the maximum performance, but this comes at a significant energy cost.

System designers have proprietary fan speed control algorithms that run in their BMCs; these attempt to minimize fan speeds while staying within the component specifications. Multiple algorithms can be used simultaneously with the final fan speed to be determined by comparingthe results of these algorithms.

Multiple (i.e., tens of) sensors are used in the algorithm with the required fan speeds set based on the components with the least margin to their specifications. The algorithm must ensure that unacceptable fan oscillations do not occur, even at low fan speeds. These could be just as annoying to a customer as a continuous loud noise.

■ **Note** It is possible for a system to transition from low-power consumption and corresponding low fan speeds to a very high-power workload in microseconds. Although the CPU die does not heat up instantaneously due to the higher power utilization, it may heat up faster than the fan speed control subsystem can increase the fan speeds, resulting in a short period of CPU thermal throttling. Fan speed control algorithms that are heavily optimized for energy efficiency can be more exposed to this type of behavior.

Fan or cooling zones are often used to precisely adjust specific fans to the needs of components most coupled with those fans. Cooling zones can be proximity based, or physically separated. The extent of optimization versus cost is considered when designing cooling segmentation created through cooling zones. Using a fan zone

www.dbooks.org

implementation enables total fan power and acoustic to be minimized. Fans in a non-stressed zone can run at lower speeds than those needed in a more highly stressed zone. A stressed zone implies that at least one component is approaching its temperature limit. Figure 4-23 shows two examples of fan zones mapped to two different boards designed for use in a 1U chassis.

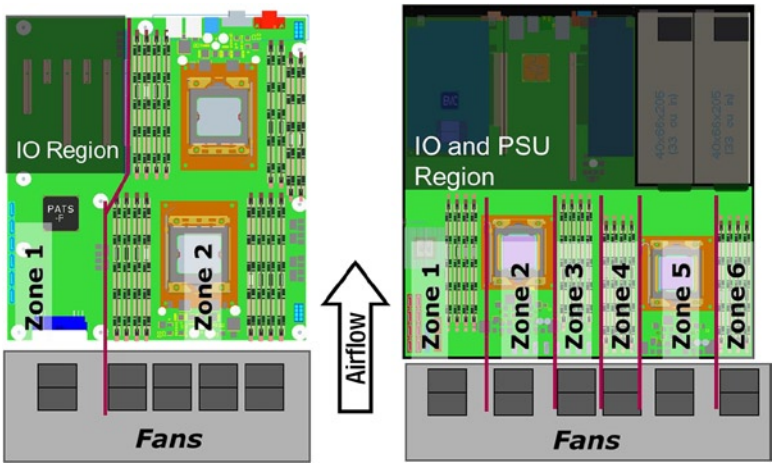


Figure 4-23. *Fan zone mapping*

Each sensor is mapped to the fan zones depending on its thermal connection to that zone. A single sensor could impact a single zone or multiple zones depending on positioning and ducting. By mapping specific components to specific fan zones, more granularity in fan control can be obtained, thereby reducing total fan power.

A variation on a proportional, integral, derivative (PID) controller is commonly used for fan speed control. For each thermal sensor or group of thermal sensors, a separate PID algorithm is running. At each time step, a new fan speed setting is determined from the PID controller using temperature value from each sensor. The management controller determines the actual fan speed setting based on the maximum calculated fan speed setting from all the simultaneously operating PID algorithms.

Fan speed settings normally have a *floor*, preventing operation below the levels necessary to cool components without sensors where the real-time temperatures are unknown.

Summary

Each CPU requires a large amount of support hardware in order to complete its tasks. Data centers are generally made up of a large of number of racks. Each rack contains a number of separate platforms (or chassis) that provide one (or a few) compute nodes. In addition to the CPUs, a platform includes the memory, drives, networking, power delivery, cooling, and more that is required for enabling a small number of CPUs to operate. Similar problems and tasks must also be managed at larger scales in a rack or even across a data center. As an example, each platform likely has dedicated cooling hardware and thermal monitoring and management capabilities. However, additional cooling is necessary for extracting heat from the rack and ultimately the data center. This topic is discussed in Chapter 9.

A wide range of platform designs are possible, with different optimization points for different usages. A storage platform may include a massive number of drives all connected to a single two-socket (DP) node. A compute node may have a single high-speed network connection, no drives, and some amount of memory that has been selected for the types of workloads that run on that node. Large EX platforms tend to have high CPU TDP powers. However, their support hardware also tends to have high costs (both for power and procurement), amortizing the cost of that additional power and making it very cost efficient. Similarly, low power offerings can be very effective if high performance is not needed.

The power/performance efficiency of a CPU is heavily dependent on the platform around it and the demands of the workload in question on that platform. Measuring performance per TDP watt of a CPU can be a very misleading statistic due to the many platform components that contribute to power. If the task that is required is bottlenecked by drives, adding more compute performance (and increasing power) will be inefficient. Similarly, if a platform includes significant power outside the CPU, and increasing the CPU power results in an increase in overall platform performance, it may be more power efficient for the platform to increase the CPU power even if the CPU performance per CPU watt decreases. Building a power efficient platform requires balancing the compute needs with the capabilities of the supporting platform devices.

www.dbooks.org

CHAPTER 5

BIOS and Management Firmware

Thousands of times a second, CPUs, memory, system interconnects, and other components transition between a number of different active and idle power states to minimize the use of energy. Energy-efficient use of these power states isn't possible without careful coordination between hardware and software—including BIOS firmware, management firmware, operating systems, and applications. If any one of these firmware or software components fails to fulfill its role in this coordination, it can cause a wide variety of problems—from increases in power to decreases in performance. Figure 5-1 illustrates the hierarchy of software components used in enabling and controlling the use of power management features.

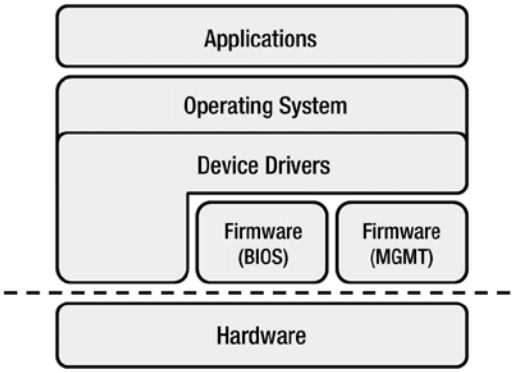


Figure 5-1. *Hierarchy of software components used in power management*

BIOS firmware is responsible for turning on and configuring power management features. BIOS must also expose power management features to the operating system (OS) to allow for software control. This advertisement includes a list of the supported power states, each state's power and performance characteristics, and a description of control interfaces. In some cases, OS device drivers can discover and configure power management features directly. The OS is responsible for monitoring the system at runtime and using the BIOS-provided interfaces to control power management features based on past, current, and projected future activity. This activity is ultimately generated by applications that use system resources to perform computations and manipulate data.

This chapter begins with a description of BIOS firmware and its role in activating and configuring features. It continues with an overview of BIOS firmware's role in updating microcode and creating Advanced Configuration and Power Interface (ACPI)) objects that describe power management capabilities to software. It includes an overview of management firmware including hardware protection, power capping, and system monitoring functions. The chapter concludes with a description of the Intelligent Platform Management Interface (IPMI) and how it is used to configure and control firmware capabilities used for power management.

BIOS Firmware

When a server is powered on, power management features such as C-states, P-states, interconnect power states, and memory power states are not configured or enabled. This is unlike many of the other system functions that do not require explicit firmware or software enabling. Enabling and configuring power management features is the responsibility of BIOS firmware.

Many power management features exist in different processor units or different system components. These features have different clock and power domains, different initialization sequences, and different enabling requirements. As a result, configuration and enabling takes place across multiple stages where BIOS firmware coordinates with the CPU's power control unit (PCU) and other units to initialize individual features. If this initialization fails, power and thermal management will not function properly.

During the various power management initialization stages, there are two common methods for communication between firmware and hardware. The first is reading and writing small data arrays in hardware called *registers*. Register size is typically measured in bits, for example, a 32-bit register or a 64-bit register. Registers provide a convenient mechanism for software and hardware communication and are used extensively in the control and configuration of power management features. The three most common types of registers used for power management are model specific registers (MSRs), control and status registers (CSRs), and memory-mapped input/output (MMIO). These registers can only be read or written in Ring 0, meaning they are only accessible to kernel mode software with the highest privilege level. Where there is a need to use a register to frequently access status information, make repeated changes to configuration settings, or regularly change power states of the processor, MSRs are used. These registers can be accessed with low software overhead using dedicated RDMSR and WRMSR instructions. Where there is power management control or status information that needs to be accessed infrequently or only during boot time, CSRs and MMIO are used. CSRs are

www.dbooks.org

registers mapped to memory locations in legacy PCI configuration space, where several layers of device drivers may be required to access these after control has been passed to the operating system.

A second communication method between software and hardware is the x86 CPUID instruction. The CPUID instruction provides a fast mechanism for software to query the processor to determine feature support and configuration information, and it is accessible in Ring 3 or by user mode software (for example, the CPU type, the topology of cores and threads, and various feature support flags). The CPUID instruction provides a low-latency and error-resilient way to determine if a processor supports power management features such as P-states or C-states, and what happens to various clock sources when the processor is idle.

Not all power management features are under direct operating system control. Chapter 2 details how Turbo allows the processor to operate above the CPU base frequency, how a multi-socket system enters a coordinated package C-state, and how power and thermal events can trigger processor throttling. These types of features are controlled by CPU microcode and PCU firmware. Activity generated by applications or OS power management policies may influence the use of these features, but the core functionality is controlled elsewhere. In addition to the PCU, other microcontrollers in the system, such as a baseboard management controller (BMC) or a Management Engine (ME) in the chipset, may also participate in the control of power management features.

Microcode Update

There may be cases where it is necessary to change the behavior of hardware power management features—to fix issues, to add new functionality, or to optimize feature use. The majority of these updates can be done through BIOS firmware updates. The greatest benefit of firmware updates is the ability to improve system behavior without having to replace any components. However, the downside is the need to restart the system in order to do so. In datacenters with tens of thousands of servers, a simple firmware update becomes an event with significant cost and complexity.

Where there is a need to enhance or correct CPU-specific behavior, this can be performed by the operating system. Operating systems include updated CPU microcode and have the ability to update CPU microcode without needing to install and validate a new BIOS firmware image or reboot the server. CPU microcode updates are done by loading microcode into memory and writing the address of the microcode to the IA32_UCODE_WRITE MSR.

Throughout this book, several of the MSRs outlined start with the IA32_ prefix. This prefix indicates that the MSR is architectural, meaning it is supported in future CPUs using a consistent address and data field definition. Data fields that are reserved or undefined can be used to add new functionality over time. Architectural MSRs also do not change across different product segments. For example, the definition of IA32_PERF_CTRL is identical across phones, tablets, notebooks, and servers. This is critical for maintaining forward and backward compatibility with the various software components utilized in power management. MSRs are documented in detail in the Intel Architecture Software Developer Manuals.

Advanced Configuration and Power Interface

As discussed in Chapter 2, C-states and P-states, or processor idle power and processor performance states, are controlled by the operating system. After these features are configured and enabled by BIOS firmware, BIOS firmware is responsible for advertising these power states and control interface information to the operating system. To accommodate compatibility and flexibility between different hardware and software implementations, an industry standard interface called Advanced Configuration and Power Interface (ACPI) is used.[1] ACPI provides an interface for conveying power state information in abstract terms so operating systems and hardware power management features can advance independently without causing any loss of functionality. For example, a new hardware C-state released in 2015 can be used efficiently by an operating system released in 2005 as long as that C-state is described using ACPI. This abstraction is necessary because there can be significant differences in power state behavior between different platforms, architectures, and CPUs, even when those power states share the same name. Similarly, OS power management policies may choose not to use a power state in one product and version, but may choose to in another. The scope of ACPI goes beyond describing power states and control interfaces. Rather than a comprehensive review of ACPI capabilities, this section discusses only those key interfaces most relevant to understanding the hardware and software interaction in power management.

ACPI provides an abstraction for several different types of states including S-states, C-states, P-states, T-states, and D-states. ACPI states are identified by a letter indicating the state type, followed by a number indicating the depth of the state. For example, S0 is system state 0 and P5 is performance state 5.

Lower numbers indicate a state with more activity and higher power—the number 0 always indicates the state with the most activity, highest performance, and highest power. As a result, lower numbered and higher power states are called *shallow power states* whereas higher numbered and lower power states are called *deep power states*. The description of states as shallow or deep is done to convey state transition costs, such as latency or the transitional energy needed to enter or exit a state. During these transitions, the system is stalled or may be taking actions that will affect performance when the processor resumes execution, such as flushing caches and translation lookaside buffers (TLBs).

A resource can only be in one state type at any given time. For example, a system can only be in one S-state at a time, a core can only be in one P-state at a time, and a device can only be in one D-state at a time. Figure 5-2 illustrates the relationships and dependencies between S-states, C-states, and P-states and how a system transitions between them.

[1] Advanced Configuration and Power Interface Specification, Revision 5.0a, November 13, 2013.

www.dbooks.org

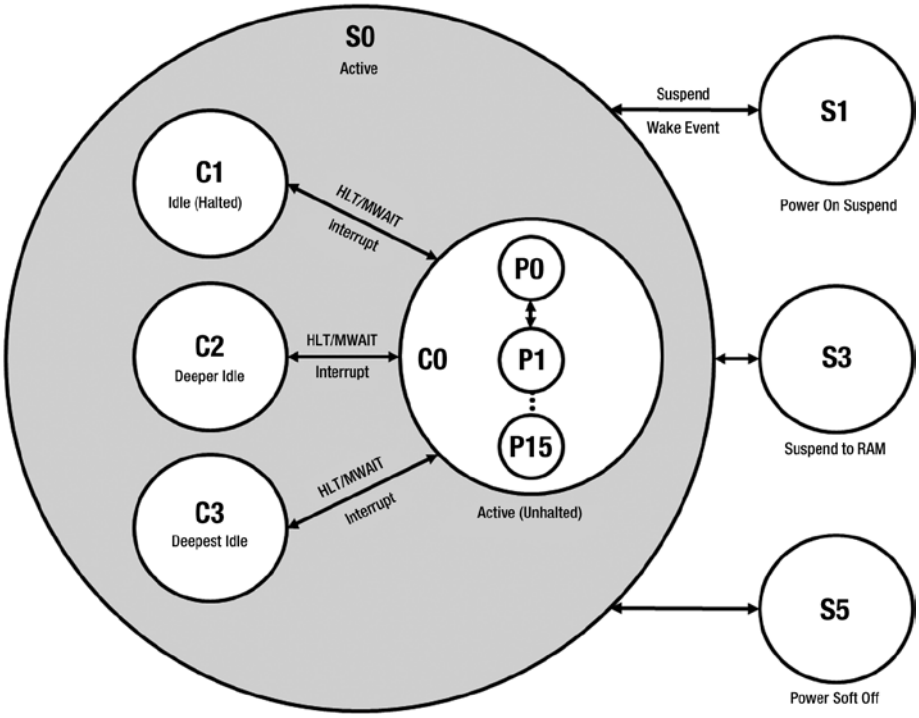


Figure 5-2. *Summary of ACPI power states and transitions*

S-states

S-states refer to system level sleep states and include S0, S1, S3, S4, and S5. The CPU only executes instructions in the S0 state. The use of other S-states is somewhat uncommon for servers, because most servers are usually in an active state (S0), where they are active or ready to execute or they are powered off (S5). A server in an S5 or soft off state is one that is powered off, but still plugged in. Even though an ACPI S0 state describes an active state, it is possible for processors, devices, or other resources in the system to be idle. System level states can be in a shallower state than processor or other device states, but they can never be in a deeper state.

ACPI S3, commonly referred to as suspend, is a sleep state where OS context is saved to system memory; memory remains powered, but most of the other system components are powered down. In S4, all devices have been powered off, but current OS context has been retained on a storage device. S3 and S4 support varies with many server products not supporting these.

■ **Note** Use of S3 and S4 is uncommon in servers. These states do not maintain an active network connection, and execution context is no longer in CPU caches. It can take a significant amount of time to resume from these states, making them difficult to use in dynamic environments with variable load.

C-states

The ACPI specification defines three types of idle power or C-states: C1, C2, and C3. A C0 state describes an active processor that is executing instructions. An ACPI C1 state is mandatory. It describes the lowest latency idle power state and is reserved for processor states that have an insignificant amount of transition latency or performance impact. An ACPI C2 state is a deeper state than C1, with lower power and higher latency. It's allowed to have measurable latency impact but does not require any additional software handling above what an ACPI C1 state requires. An ACPI C3 state is the lowest power and highest latency state and has extra software overhead associated with C-state entry and exit. ACPI C3-type states have software visible effects. Use of these states may require the OS to check on chipset activity before entering the state or may require the OS to identify and use alternative time sources due to a processor timestamp counter or local APIC timer stopping after entering the C-state.

Due to the increased software complexity of ACPI C3-type states, most modern servers do not implement C-states that map to anything deeper than an ACPI C2-type state. Over time, hardware C-states have been optimized to eliminate or reduce software visible effects. This ranges from architecting timers so they continue to run when the processor is in deep C-states and eliminating dependencies on activity level outside the CPU to aggressively reducing deep C-state exit latencies.

■ **Note** A common point of confusion is the difference between hardware C-states and APCI C-state types. Each of the hardware C-states described in Chapter 2 is mapped to an ACPI C-state type when they are advertised by BIOS firmware. For example, hardware C1 states map to an ACPI C1 type whereas hardware C3 and hardware C6 states map to an ACPI C2-type state.

P-states

P-states refer to processor performance states and include ACPI P0, and P1 to P*n*, where the number of states between 0 and *n* varies based on the number of unique voltage and frequency operating points supported by the processor. P*n* is also referred to as the deepest P-state. Unlike S-states or C-states, which represent idle states, P-states represent active states, and as a result, ACPI P-states are only utilized when the processor is in C0, actively executing instructions.

www.dbooks.org

The P0 state is the highest performance and highest power state, and every state from P1 down to P*n* results in a decrease in power and a decrease in performance in comparison to lower-numbered states. ACPI limits the number of P-states to no more than 16. In cases where a processor has more than 16 hardware P-states, BIOS firmware must decide which of these are exposed to the OS. BIOS firmware typically exposes an ACPI P-state for every base clock step between the processor's minimum frequency (P*n*) and the CPU base frequency. Turbo mode, discussed in Chapter 2, is always mapped to ACPI P0.

D-states

The ACPI specification also defines device power states, or D-states. ACPI D-states aren't utilized as frequently on servers as they are in clients. Many servers are unable to use D-states since the latency to resume from these states is too significant for use in active servers. Another reason ACPI D-states aren't always exposed on servers is because additional standard interfaces for device state discovery and control exist, such as Power Management Control and Status register (PMCSR) , defined by the PCI and PCI express specification. Some device drivers manage native device-specific power states via private device-specific controls. Many devices have the capability to monitor their own device activity and manage power without any software control. Even where there are no software exposed D-states, devices or CPUs may be autonomously transitioning between various power states at runtime to manage power.

Although there are various control methods and specifications that describe device power management outside of ACPI, they all share the same terminology. A D0 state is active, D1 and D2 are low-power states where device context is saved, and D3 is the lowest power, highest latency state where no device context is saved.

ACPI Interfaces

An operating system needs a much greater level of detail about power states and their behavior in order to utilize them efficiently. When selecting between different power states, the OS needs to know each state's power consumption, the transition time to enter and exit a state, what level of execution context is lost upon entering a state, and specific mechanisms for initiating entry. ACPI standardizes tables to describe this detailed information to the OS.

BIOS firmware is responsible for constructing these tables and loading them into memory where the operating system reads them and enumerates capabilities. ACPI tables that include core power state information are the DSDT (Differentiated System Description Table) and the SSDT (Secondary System Description Table). These tables consist of several objects that provide needed power state and control information to the OS. The following list describes the primary set of ACPI objects used by the OS:

- **_OSC and _PDC (Operating System Capabilities and Processor Driver Capabilities):** These methods are used to communicate capabilities of the OS to BIOS firmware. This includes describing OS capabilities in terms of coordinating control across multiple logical and physical processors, and its ability to control P-states and C-states. The capabilities of the OS will determine what power management features BIOS firmware will expose.

- **_PSS (Performance Supported States):** This object lists the P-states available to the OS. For each state, the object includes a performance level (typically core frequency in MHz), the maximum power consumption, and transition latency. In addition, the object lists a control register value that the OS uses to identify a P-state when requesting a power state change and a status value that the OS uses to identify a P-state when checking on processor status. A separate ACPI object defines the processor status register that the OS uses to determine the current P-state and to check the status of existing P-state control requests.

  The ACPI specification describes P-states in terms of guaranteed frequency. Turbo mode, or the ability for processor cores to run above the CPU base frequency, is not guaranteed. Turbo is opportunistic with the frequency dependent on thermal or power headroom. Since Turbo frequency is not guaranteed, it is improper to expose the maximum Turbo frequency via ACPI. As a result, the ACPI _PSS exposes Turbo (P0) at 1 MHz higher frequency than P1. When the OS requests Turbo, hardware will maximize frequency, potentially running well above what is advertised to the OS. If frequency determinism is a hard requirement for users, software interfaces are provided so Turbo can be disabled.

■ **Note** There are many cases where exposing Turbo as a single P-state is not energy efficient. Not allowing the operating system to choose intermediate P-states between P1 and the maximum Turbo frequency can result in selection of a P-state that is higher performance and power than required.

- **_PSD (P-state Domain):** The ACPI _PSD object describes CPU control dependency, defining whether logical processors in a CPU have their own P-states or whether subsets of logical processors share a common P-state. Some server CPUs have a single P-state domain, meaning all logical processors in that CPU share a single frequency. Other server CPUs have a different P-state domain for each core, meaning that all cores in the CPU can run at their own independent frequency. The _PSD object is also responsible for describing the P-state coordination type, which is discussed in greater detail later in this chapter.

www.dbooks.org

■ **Note** Errors in ACPI objects can cause significant power issues. Figure 5-3 shows the impact of an online transaction processing workload running on a system with an incorrect _PSD. These issues have resulted in substantial (up to 40 W higher) power increases throughout a range of different operating conditions.

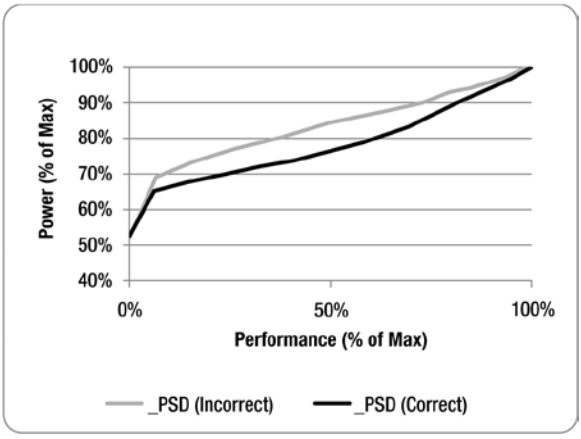


Figure 5-3. *Power impact from ACPI _PSD object with an incorrect mapping*

- **_PCT (Performance Control):** This object describes the processor registers or firmware locations that allow the OS to change P-states and to check the status of P-state requests. To change P-states, the OS uses the _PCT-specified control register. P-state transitions are initiated by the OS writing a P-state's control value (specified in the _PSS) to the control register. In order to check the status of P-state requests, the OS uses the _PCT-specified status register. The status register also specifies the current P-state in terms of the _PSS-specified P-state control value.

  Most modern processors specify these interfaces in terms of what ACPI calls Functional Fixed Hardware (FFH) , or a processor MSR. This allows the performance control interface to be implemented directly in hardware providing a low latency and error resilient interface. Where a native processor interface is not available or desirable, an original equipment manufacturer (OEM) can implement platform-specific code to handle performance control.

- **_CST (C-states):** This object lists the C-states available to the OS. Details provided for each C-state include the register used to place a processor into a C-state, the ACPI C-state type, worst-case latency, and typical power consumption. Power consumption numbers in the _CST are not used by operating systems because they are assumed to be estimates only. The latency field is used by several OS control policies to limit use of some C-states based on system activity levels or where there is a specific device that can't tolerate latency above some threshold.

  Similar to ACPI P-state objects, an ACPI _CSD object exists to describe processor control dependences for C-states. Understanding cross logical-processor C-state dependencies is useful for understanding C-state impact when the OS needs to consolidate execution to some subset of logical processors in the CPU.

After an operating system initializes, it evaluates these ACPI methods and has all the information it needs to request hardware transitions between various idle and active states and to check the status of those requests. The OS control policy uses ACPI-advertised information outlining the expected power and performance impact of the various states to make state transition decisions, and it uses ACPI-advertised control mechanisms to execute those decisions.

■ **Note** There is a long history of issues with the resiliency and robustness of the ACPI interface for OS power management. Some modern operating systems are starting to use native processor interfaces such as CPUID to discover the CPU type, power management features, and control interfaces directly from hardware. This use of native processor interfaces limits how flexible the power management solution is, but it eliminates errors in ACPI objects from causing functional or performance issues.

Setup Utility

The setup utility, also implemented in BIOS firmware, is a powerful tool for fine-tuning power management and optimizing a platform for specific workloads. The majority of power management features can be enabled, disabled, or have their default behavior changed through simple setup options. Chapter 8 is a comprehensive optimization reference that discusses options commonly found in the setup utility and different ways these options can be configured to decrease power, increase performance, or, in an ideal scenario, both.

www.dbooks.org

Management Firmware

Microcontrollers in cars can monitor fuel level and consumption rate. They can indicate if tire pressure is low or if a turn signal is burned out, and they can keep a record of diagnostic events that can be retrieved by a technician during a service appointment. These capabilities are provided independent of who is driving the car, or if the car is speeding or is stopped.

Management firmware running on server microcontrollers plays a very similar role. Management firmware provides power, monitoring, event logging, inventory, and remote management capabilities independent of the OS or state of the processors. This is particularly useful in the datacenter where there is large number of servers, where systems are going up and down for maintenance, and where servers are running different operating systems. Two key firmware components that are critical for power management are the baseboard management controller (BMC) firmware and the Management Engine (ME) firmware, called Node Manager.

Node Manager Capabilities

Node Manager firmware provides key capabilities for managing and optimizing both power and cooling resources in the datacenter. It exposes a standardized set of hardware protection, monitoring, and power capping features to the BMC and to external management software. Node Manager acts as a satellite controller and offloads power management responsibilities from the BMC, with some of the capabilities always running and others activated by a profile.

Hardware Protection

Node Manager firmware implements a set of hardware protection mechanisms to protect the platform during adverse or unexpected conditions. There are proactive protection mechanisms such as dynamically limiting platform power to the capabilities of the PSU. There are also reactive protection mechanisms such as closed loop system protection (CLST) and Smart Ride Through (SmaRT) that protect the platform during PSU over-temperature, under-voltage, and over-current events. These capabilities are hardware assisted, with sensor devices using the SMBUS protocol to notify Node Manager about critical events. Protection mechanisms respond immediately in the case of an adverse condition, with required actions, such as processor and memory throttling, occurring in under a millisecond.

Monitoring

Another key capability of Node Manager firmware is comprehensive platform monitoring. In addition to the monitoring capabilities provided by processors and memory, modern servers implement several onboard sensors in intelligent power supply units (PSUs), voltage regulators (VRs), hot swap controllers (HSCs), and in devices accessible by the BMC. These sensors enable fine-grained power monitoring since they are capable of reporting voltage, current, power, and energy consumption for individual components. Combining together all the board, processor, and memory sensors creates a sensor grid that Node Manager firmware relies on for power management.

These monitoring capabilities have uses beyond enabling Node Manager's protection and power capping features. External management software uses these monitoring capabilities in a variety of ways. For example, events that monitor inlet temperature, outlet temperature, and volumetric airflow are used by facility control software.

■ **Note** In order to expose more information about the platform, Node Manager adds several synthetic sensors such as outlet temperature and volumetric airflow. These sensors are derived from other sensors in the platform and calculated based on a mathematical model.

Events that monitor compute utilization and memory utilization are used by orchestration software to aid in workload placement and migration decisions. Events that monitor power consumption are used to characterize and optimize production workloads. Various types of sensors and usages are described in greater detail in Chapter 7.

Power Capping

Node Manager firmware allows users to set and enforce a power cap ensuring that power will not exceed a defined threshold. External management software uses this capability in several different scenarios to provide power, performance, and cost benefits.

Most servers operate well below the theoretical maximum platform power, even when workloads are running at peak throughput. This limits the number of servers that can be safely added to a rack with fixed power capacity. Rather than allowing a server to operate up to the theoretical maximum platform power, users can enforce a power cap that corresponds to more representative peak conditions. This power cap can be determined using insight gained from datacenter monitoring, it can be established by characterizing production workloads under peak conditions, or it can be established based on some percentage of the theoretical maximum platform power. Using a more representative power cap, rack density can be improved.

■ **Note** Node Manager provides a feature that automatically characterizes platform minimum and maximum power during BIOS POST using specialty workloads. The results of this characterization can be used to identify an appropriate power cap when it is not possible to do so using production workloads.

Several additional applications of power capping exist—for example, power capping to survive a power or cooling failure in the datacenter. An aggressive power cap can be enforced during these conditions, decreasing server power and cooling requirements. This keeps applications running, delaying or avoiding automatic shutdown. Power capping can also be applied strategically to maximize resources where energy has a variable cost. Figure 5-4 shows external interfaces and components used by Node Manager firmware to enable hardware protection, monitoring, and power capping.

www.dbooks.org

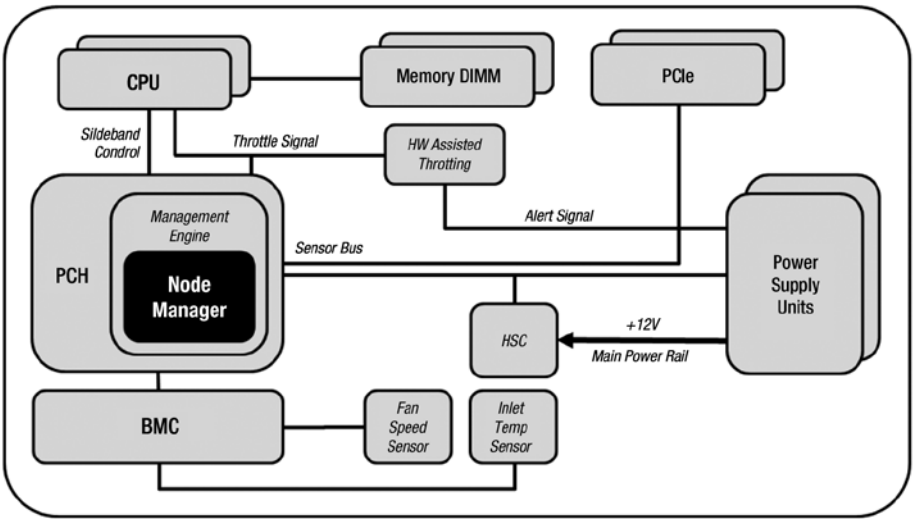


Figure 5-4. *External interfaces and components used by Node Manager firmware*

Node Manager Policies

By default, Node Manager firmware activates hardware protection and basic monitoring. Additional Node Manager firmware capabilities are supported using user-defined policies that can be created, configured, enabled, and disabled.

Policies can be always running, such as a power capping policy that replaces the theoretical maximum platform power with a cap more representative of peak conditions. Policies can also be enabled by a trigger, or some monitoring event used to activate the policy. For example, a policy can enforce a power cap only when an inlet temperature exceeds some threshold. An operator might define several different inlet temperature thresholds, with each one activating a different power cap.

Node Manager policies typically monitor or control power for a specific policy domain. A domain is simply an abstraction for related individual platform components. For example, a policy that targets the platform domain includes all components in the server. The CPU domain would report and control power for all CPUs in the system, treating them as a single entity, while the memory domain reports and controls power for all DIMMs and memory controllers in the system.

Table 5-1 lists all the different attributes of a Node Manager policy that operators can use to configure policies to match desired behavior and specific needs. These attributes enable more sophisticated event-driven management. For example, if the server is unable to meet a power cap, the policy can define a resulting action, such as sending an event to external management software or shutting down the platform.

Table 5-1. *Attributes of a Node Manager Policy*

Attribute	Supported Values	Description
Assigned policy ID	A one-byte numeric value.	Indicates a unique identifier for the policy. This is assigned during policy creation.
Policy domain	Can be any of the following: • Entire platform • CPU subsystem domain • Memory subsystem domain • Hardware protection domain • High-power I/O domain	Indicates the specific platform subsystem the policy is applied to.
Administrative state for policy	Can be either of the following: • Enabled • Disabled	Indicates the state of the policy. Even if a policy is disabled, monitoring for the policy is still enabled.
Policy trigger type	Can be any of the following: • No policy trigger. • Inlet temperature limit (in Celsius). • Missing power reading timeout (in 1/10th of a second). • Time after host reset (in 1/10th of a second). • Boot time policy. This policy will be applied only at boot.	Indicates the trigger for policy activation. If "No policy trigger" is specified, the policy is always active. For all other triggers, the policy is only active while the condition is true.
Policy trigger limit	A temperature value in Celsius or a time value in 1/10 of a second.	Indicates the specific value associated with the trigger.
Policy limit	A power cap can be specified as one of the following: • Power (in W) • Throttling level (in %)	Indicates a power cap to be enforced. Platform throttling level is used in case of missing power readings.

(*continued*)

www.dbooks.org

Table 5-1. (*continued*)

Attribute	Supported Values	Description
Aggressiveness	Can be one of the following: • Automatic • Force unaggressive mode • Force aggressive mode	Indicates the types of power management mechanisms used to keep the server below a power cap. Node Manager attempts to meet a cap using the most energy efficient mechanism available. Mechanisms with greater performance impact are used only when a cap cannot be met using the energy efficient mechanisms.
Correction time limit	A time value in milliseconds.	Indicates the maximum time, in milliseconds, in which the Node Manager must take corrective actions to meet a power cap. If this time is exceeded, the "Policy exception action" specifies the next action.
Policy exception actions	Can be either or both of the following: • Send alert. • Shut down system (hard shutdown via BMC).	Indicates action taken if the policy limit cannot be met. Sending an alert will cause Node Manager to generate an asynchronous event to notify external management software that the defined limit is too low.
Policy storage option	Can be either of the following: • Persistent storage • Volatile memory storage	Indicates the storage type of a policy. By default the policies are stored persistently so Node Manager will restore the policies after each platform reset. If policies are frequently created and updated, volatile storage should be used.

(*continued*)

Table 5-1. (*continued*)

Attribute	Supported Values	Description
Statistics reporting period	A time window in seconds.	Indicates the averaging window for monitoring. This allows operators to specify up to a one hour moving average window for monitoring.
Alert thresholds	Up to three thresholds in the units specified by trigger type. For a policy without a trigger, the thresholds array contains average power in watts. For temperature-based triggers, the thresholds array contains temperature in degrees Celsius. For time-based triggers, the array contains time in 1/10 of a second.	Indicates threshold trigger values need to exceed to generate events.
Suspend periods	An array of start and stop times including recurrence patterns based on the day of the week.	Indicates when the policy will be enforced.

The policy allows operators to specify various alert thresholds. Each policy supports up to three thresholds that can be used to generate events. For example, it is common to set a threshold close to the defined power cap, so external management software can see how close the system is getting to enforcing a cap.

The policy allows operators to specify suspend periods. This defines a weekly pattern of days and times a policy should be enabled or disabled—for example, power capping servers hosting IT infrastructure during nights and weekends. Node Manager automatically synchronizes the real-time clock used for scheduling with the host OS real-time clock to keep software and systems in sync.

IPMI

BMC and Node Manager firmware capabilities are configured and controlled through the Intelligent Platform Management Interface (IPMI).[2] Support for IPMI is widespread, with the vast majority of servers supporting it. IPMI provides a standard well-defined interface between external management software and the underlying platform, enabling various monitoring, logging, inventory, and recovery functions using simple request and response messages.

[2]IPMI Specification, v2.0, Rev. 1.1.

www.dbooks.org

IPMI messages or commands target the BMC. The BMC acts as a communication hub for satellite controllers in the platform that include their own monitoring and control capabilities such as the ME in the PCH. Communication between the BMC and satellite management controllers takes place over an I2C bus using the Intelligent Platform Management Bridge (IPMB) interface. The I2C bus, SMBus, PMBus, memory-mapped I/O ports, as well as private management busses are all used to connect management controllers to various sensors in the platform.

Use of IPMI eliminates the need for vendor-specific tools that are incompatible between different platforms. Since IPMI is an open standard, it also allows servers to implement management functionality independent of the OS, BIOS, or the system configuration. Monitoring functions in the BMC can be accessed by IPMI out-of-band, over the network by a connected client. These functions can also be accessed by IPMI in-band through management tools and device drivers installed on the server.

Sensor Model

Sensors in a platform, such as a CPU and memory temperature sensors are discovered by management software using IPMI commands. This discovery is aided by the IPMI sensor model. The sensor model describes all the different sensors supported, as well as each sensor's name, type, and the values they return. Some sensors may provide real-time measurements whereas others may provide only a count or indication of past events.

Sensor information is stored in IPMI sensor data records (SDRs). In addition to storing information on sensor capabilities, the SDR is used to describe the various devices connected to the ICMB and it associates each sensor with the host management controller. The SDR also provides information on event generation capabilities and describes thresholds that can be set to trigger events.

■ **Note** IPMI is extensible so server manufacturers are able to add their own custom sensors and commands. As a result, management controller monitoring capabilities can vary greatly from one server to another.

Inventory information such as FRU (field replaceable unit) devices connected to the platform are stored in the SDR. FRU data includes information for inventory management such as serial number, part number, manufacturer, and description.

System Event Log

Events generated by the BMC and by Node Manager firmware are stored in a centralized event log called the System Event Log (SEL)—for example, an indication that a fan is no longer functioning properly. Similar to SDR access, IPMI enables access to the SEL providing common functions such as reading or clearing the log.

Satellite controllers send their messages to this centralized log via the IPMB, allowing the SEL to act as a single platform event repository. Stored in flash memory, events captured in the SEL contain critical information that isn't lost if power is disconnected or there is an operating system failure.

Node Manager API

Node Manager capabilities described earlier in the chapter are accessed and controlled through IPMI commands. The Node Manager IPMI API includes numerous IPMI commands for creating policies, configuring policies, accessing monitoring capabilities, and accessing runtime attributes.

The Node Manager IPMI API describes commands operators can use to query capability and version information. External management software relies on this interface to discover capabilities as different platforms may expose a different set of policy domains and features.

The API describes commands operators can use to specify the destination of alerts. Node Manager defines a set of events that are sent directly to external management software, bypassing the BMC SEL. For these events, the IPMI Alert Immediate API is used. This gives external management software the choice between event-driven or periodic polling management. The API also includes commands that provide additional management functionality. For example, commands that enable operators to set a Turbo synchronization ratio, or a frequency limit for Turbo. This feature can be used to improve performance determinism in high performance computing (HPC) environments.

A complete list of commands included in the Node Manager IPMI API are included in the Node Manager specification at www.intel.com/content/www/us/en/power-management/intelligent-power-node-manager-specification.html.

The Node Manager IPMI API includes two types of interfaces:

- **External API:** This is designed for use by external management software. This API uses the policy domain abstraction to expose high-level monitoring and control to external management software. Exposing platform management capabilities at a domain level simplifies management and improves scalability, especially in environments with thousands of servers to manage.
- **Internal API:** This is designed for use by the BMC or other management controllers in the system. This low-level API allows the BMC to access specific sensors and control features as needed.

www.dbooks.org

ACPI Power Metering Objects

In addition to IPMI, Node Manager also exposes power monitoring and power capping capabilities through ACPI power metering objects. This allows the OS to participate in monitoring and control. The key ACPI objects that enable this functionality are outlined here:

- **_PMC (Power Meter Capabilities):** This object describes the power meter capabilities including measurement unit, type, accuracy, and sampling time. It describes whether the platform is capable of monitoring platform power, enforcing a power cap, or both.
- **_PMM (Power Meter Measurement) and _PAI (Power Averaging Interval):** The _PMM object returns the latest reading from the power meter. The averaging window for the _PMM returned reading is defined by the _PAI.
- **_PTP (Power Trip Points):** This object is used to define the upper and lower trip points for the power meter.
- **_SHL (Set Hardware Limit) and _GHL (Get Hardware Limit):** These objects are used to enforce a platform power limit.

Summary

BIOS and management firmware play a critical role initializing, configuring, controlling, and advertising power management features to external software. Without these actions, systems would fail to utilize power management features, resulting in high power usage, low performance, or both. Standard interfaces such as ACPI and IPMI are used to enable firmware to communicate with external software in an OS-independent fashion.

Chapter 6 will continue the discussion on software architecture with a description of the operating system's role in power management. It will describe how the OS uses BIOS firmware exposed ACPI objects to enumerate and control various power states. It will also describe how OS power state selection, process scheduling, and memory management decisions made by the OS impact energy efficiency.

CHAPTER 6

Operating Systems

Software computation and data manipulation is the essence of the work done on a server and in a datacenter. What is often overlooked is the critical role operating systems play in determining both the performance (the speed of work) and the cost (the energy consumed) doing that work. Operating systems manage a software work plan; an efficient work plan gets work done faster and at lower cost.

Well-optimized operating systems can increase the time spent in low-power CPU, memory, and interconnect states. They avoid careless use of system resources that can limit a server's use of low-power states—for example, polling for completion of some event or activity, choosing a suboptimal resolution of the system timer, or performing operations that only utilize a small subset of logical processors in the system. Well-optimized operating systems adapt their use of power management features to match changes in workloads, changes in server utilization, and changes in operator preferences.

This chapter begins with an overview of operating systems and their role in selecting power states, scheduling, and memory management. This includes a description of the interfaces operating systems use for power state control as well as the policies and metrics the OS uses to drive those decisions. Additional considerations for virtualized environments are discussed, including consolidation and virtual machine migration. The chapter concludes with a comparison of different operating environments, the unique power management capabilities of these environments, and how these capabilities have changed over time.

Note The term *operating system* (OS) used throughout this chapter is inclusive of both a native operating system and a virtual machine monitor (VMM). A VMM assumes the same power management roles and responsibilities that a native OS does, so information in this chapter applies equally to both environments.

---

www.dbooks.org

Operating Systems

Once equipped with power state information and the appropriate controls from BIOS, the OS must implement power management policies that determine how and when to use various power states based on system activity or any known power and performance constraints. Striking the right balance between low power and high performance is a difficult problem since the right balance differs greatly from environment to environment and user to user. Operating system power management (OSPM) policies may be unaware of service-level agreements in place or of quality of service requirements. They may not understand whether the system is running a stand-alone application or whether it is part of a complex distributed application. Even knowledge of specific applications executing tells the OS very little about what the appropriate balance is between power and performance. To an OS, a web server hosting the front end for a family photo sharing service looks identical to a web server hosting the front end for critical financial transactions.

The diversity of power and performance requirements in a datacenter presents a unique challenge for implementing and optimizing OSPM policy on servers. In other compute environments, such as phones and tablets, common performance expectations are known ahead of time. For example, gaming is good at 30 frames per second and great at 60 frames per second. Holding a device in your hand shouldn't make your hand perspire, touch gestures should be processed in a matter of milliseconds, and creating an audible pause during audio playback is unacceptable.

In a server environment, the performance impact of power management features varies greatly based on the workload, application, system configuration, and performance requirements. Some workloads will see no performance impact whereas others may see substantial impact. In the absence of known performance requirements, OSPM policy needs to make the best decisions it can regarding the balance between low power and high performance using the limited information it does have. The two primary policies the OS is responsible for is the selection of C-states and P-states. The following sections describe the mechanisms these policies use for hardware state control as well as monitoring capabilities and metrics used for policy feedback and decision making.

C-state Control

As indicated in Figure 6-1, most operating systems implement OSPM policies in device drivers. The processor can be thought of as just another device in this context. The drivers communicate with hardware using a set of standard control interfaces that are described by ACPI or ascertained with CPUID. When an OS has no work to be done on a logical processor, it enters an idle function where eventually one of two instructions can be executed to transition the logical processor into an idle C-state.

ACPI c-state	Hardware C-state	Number of Sub-states	Description	ECX [bit 0]	EAX [bits 7:4]	EAX [bits 3:0]
				Treat interrupt as break event even if masked?	Target State?	Targrt Sub-state?
C1	C0	1	Skip MWAIT	0/1	1111	0
C1	C1	2	C1 - Halt execution	0/1	0	0
C1			C1E - Halt execution, then drop core voltage and frequency	0/1	0	1
C2	C3	1	C3 - Halt execution, flush core caches	0/1	1	0
C2	C6	1	C6 - Halt execution, flush core caches, power gate core	0/1	10	0

Figure 6-1. *Example MWAIT hints OS uses to specify various processor C-states*

MWAIT

The most common way for an OS to enter an idle state is by executing the MWAIT instruction. MWAIT can only be executed in Ring 0, so applications are unable to directly enter idle states. MWAIT provides the OS control over the specific C-state and sub-state to enter and under what conditions that C-state can be exited. The OS conveys C-state control details through two general-purpose registers used by MWAIT.

The OS uses the ECX register to specify how it wants C-states to exit upon external interrupts. When the OS-specified interrupts should be treated as break events, execution resumes within the idle function rather than directly executing an interrupt service routine (ISR). The OS uses the EAX register to specify the target C-state (C1, C3, or C6) and the target sub-state. The MWAIT sub-state is used for lower-level control of a specific C-state. For example, if a specific C-state flushes the caches upon C-state entry, different sub-states may be used to modify the behavior of the cache flush, such as flushing the entire cache at once, or flushing the cache progressively over multiple time-delayed phases. Figure 6-1 is an example of various C-states supported on a processor and the MWAIT hints used to specify these states.

Any interrupt—such as a timer interrupt, device interrupt, or inter-processor interrupt (IPI)—will cause a C-state to exit and execution to resume. In addition to C-states exiting based on interrupts, MWAIT can be used in combination with a MONITOR instruction. Executed by the OS before the MWAIT instruction, a MONITOR instruction allows the OS to specify a memory address for the processor to monitor. With a MONITOR instruction armed, any subsequently entered C-state will be exited if the monitored data address is modified. This provides an additional mechanism for software defined C-state exit conditions. For example, the Linux kernel utilizes this mechanism by arming MONITOR/WAIT pairs with a kernel `need_resched` flag. This flag is modified by the OS to indicate that the kernel scheduler should be activated on a particular logical processor. Exiting C-states based on a modified MONITOR address also provides a mechanism for waking up a specific logical processor without needing to generate an IPI.

www.dbooks.org

HLT

Older operating systems, or operating systems with deep C-states disabled, may execute the HLT (halt) instruction in place of an MWAIT in an idle function. HLT does not require, nor does it support, the same level of control that MWAIT does. HLT simply places the logical processor in the shallowest state, or the hardware C1 state. With the introduction of MWAIT, there is no longer a need for HLT. In modern systems, microcode converts HLT instructions into MWAIT instructions requesting hardware C1.

C-state Policy

In simple terms, the role of OSPM C-state policy is to predict the future. Shallow C-states are best used when the system has very short idle durations, for example, packet processing over a high-speed network. Deep C-states are best used when the system has long, uninterrupted idle durations; for example, idle time between submitting and completing an I/O. Figure 6-2 shows sample utilization from a single logical processor. It shows a mix of different shorter and longer idle durations, and ideal use of C-states.

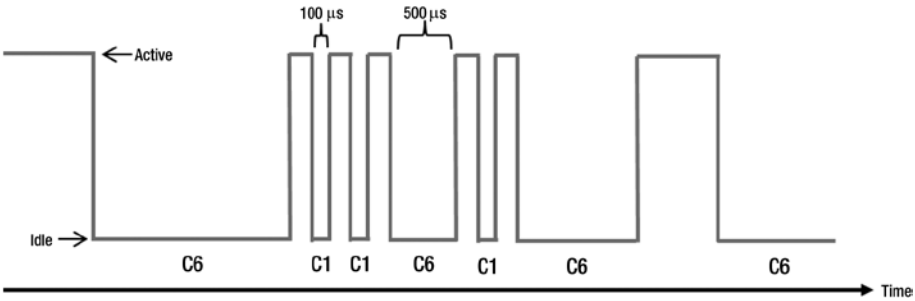


Figure 6-2. *Sample logical processor utilization with requested C-state*

With each logical processor generating unique utilization characteristics, OSPM policy needs to make individual decisions for each of them. Hardware is responsible for coordinating logical processor-level requests from two logical processors to determine the core C-state. For example, if one logical processor on a core is requesting C6 and the other is requesting C1, the core will go to C1. Similarly, hardware coordinates between core C-states to determine the package C-state. During hardware coordination, the shallowest of all C-state requests is used.

Optimal use of C-states is important for both power and performance. As discussed earlier, deep C-states introduce a performance penalty when they are exited. Using a deep C-state throughout a sequence of short idle periods results in a significant amount of stall time relative to the time spent executing instructions. In addition to latency penalties, there is also a transitional energy cost associated with C-state entry and exit. It may be counterintuitive, but using deep C-states throughout a sequence of short idle durations may actually result in higher power and lower performance. An idle duration has to be up to hundreds of microseconds long for the deepest C-states to save power—otherwise the energy cost of entry and exit has not been amortized.

C-state latency information exposed by ACPI is of limited use to OSPM C-state policy. It can be useful where software knows exactly what latency it can and can't tolerate. However, the latency value exposed by ACPI represents worst case latency, a value that can be several times greater than average or typical latency and is encountered only when in a package C-state. Cases where C-state latency is equal to the ACPI exposed value may never be encountered. In reality, there is no single value that represents C-state exit latency—it is variable and heavily dependent on other CPU activities. An additional downside of ACPI exposed latency is that it does not convey other performance visible effects such as cache and TLB flushes. These actions momentarily slow execution when the processor wakes up from a C-state due to running on caches that no longer contain recently used instructions and data.

C-state power consumption information exposed by ACPI also provides an incomplete picture from a power perspective. The power consumption of a logical processor, core, and package C-state are all very different, but ACPI objects expose only a single power value. In addition, ACPI C-state objects don't capture the idle duration necessary to break even from an energy perspective. In general, trying to convey to software accurate power consumption for some low-level state is problematic. Actual measured power depends on a number of external factors such as part to part CPU variation, temperature, and the efficiency power delivery components such as VRs and power supplies.

Chapter 2 discusses hardware C-state demotion mechanisms that processors use to prohibit deep C-states. If the PCU detects that such C-states are being used, the result may be a measurable performance impact or power increase. With these features in place, the OSPM C-state policies have the option of simply specifying the deepest state they tolerate, instead of trying to determine an ideal state, and allowing for further C-state selection optimization in hardware. This option has the benefit of eliminating complex software algorithms and simplifying the path to entering idle.

Processor Utilization

One metric OSPM C-state policy uses to determine whether to use deep or shallow C-states is *processor utilization*. This metric describes the percentage of time the processor is active or unhalted. The CPU provides fixed counters the OS can use to measure utilization on each logical processor. For example, the IA32_FIXED_CTR1 MSR described in Table 6-1 counts unhalted cycles at the rate of CPU base frequency (or P1 frequency) and the time stamp counter (TSC) MSR described in Table 6-2 counts all cycles, both halted and unhalted, at the same rate of CPU base frequency. Measuring the two events over some time period allows the operating system to calculate average utilization—the ratio of unhalted cycles in comparison to all possible cycles.

Table 6-1. *IA32_FIXED_CTR1 MSR (0x30B)*

Bits	Width	Field	Description
63:0	64 bits	Unhalted cycles	Always running measure of logical processor utilization. Increments when processor is active at the CPU base frequency of the part.

www.dbooks.org

Table 6-2. *IA32_TSC MSR (0x10)*

Bits	Width	Field	Description
63:0	64 bits	Timestamp	Always running measure of logical processor time. Increments when the processor is active or idle at the CPU base frequency of the part.

In reality, a logical processor can only be in one of two utilization states at any given time, active (100%), or idle (0%). It's observing and averaging utilization of a logical processor over some longer time window that gives us the concept of being partially utilized.

Each OSPM policy has to decide over what length of time it will observe utilization. If the time window is too short, it's likely the average utilization result will be 0%, 100%, or something very close to those endpoints. If the time window is too long, the OS will fail to identify fleeting changes or other interesting characteristics in the variation of utilization. In addition to observing utilization over a single time window, OSPM may use multiple time windows to gain additional insight into longer term utilization trends. Another OS metric used for selection of C-states is known future activity, such as periodic timers that are due to expire soon, I/O outstanding, or repeating patterns of device interrupts. Unlike utilization, these methods rely on known events and can be predicted more accurately.

The complexity in OSPM accurately predicting idle durations, trends, or patterns in idle, and the potential for error in these predictions, does not prevent a good decision from being made. In fact, appropriate use of C-states can be realized with very simple C-state policies. In modern processors, deep C-state exit latencies are only a fraction of what they were when the technology was first introduced, so the impact of a suboptimal decision has been reduced significantly. Although there are environments with acute latency sensitivity, this is the exception rather than the norm. The majority of datacenter workloads see no measurable performance impact from the use of C-states.

P-state Control

P-states represent the most significant power performance tradeoff in a server today. In most systems, the power and performance impact of P-states is greater than the impact of all other power management features in the platform combined. The impact of OSPM P-state decisions is substantial. On a multi-socket system, a given decision could result in 100 watts lower power or cutting transaction response times in half. As such, OSPM P-state policies must identify a balance between low power and high performance that meets most of their user's needs. It's impossible to meet all users' requirements, so OSPM policies must also provide mechanisms for administrators to customize OSPM behavior to meet their needs.

Similar to C-states, software use of P-states has been greatly simplified over time. For example, early P-state implementations required the processor to be in a coordinated idle state, holding off system activity through the completion of a P-state transition. In modern processors, P-state transitions are low latency and are performed dynamically.

Software Controlled Interface

Software utilizes writes to MSRs rather than executing instructions to initiate P-state transitions. State transitions are initiated by OSPM writing a control value (or frequency ratio relative to base clock) to the IA32_PERF_CTL register. Table 6-3 describes this interface and its capabilities. The ACPI _PSS object discussed in Chapter 5 describes each individual P-state based on the control value. It acts as an identifier that hardware associates with an internal frequency and operating point.

Table 6-3. *IA32_PERF_CTL MSR (0x199)*

Bits	Width	Field	Description
7:0	8 bits	Reserved	Unused
15:8	8 bits	P-state target	Control value (frequency ratio relative to base clock) used to transition logical processor to the P-state target
31:16	16 bits	Reserved	Unused
33:32	1 bit	Turbo mode disable	Disables Turbo mode
63:34	31 bits	Reserved	Unused

IA32_PERF_CTL is accessible by the OS, by firmware, and by advanced applications with kernel-level privileges. There are cases where multiple entities may try to control P-states simultaneously. Although these cases are rare, where they exist, the administrator must take great care to ensure the capability is disabled for entities that should not have control.

The ACPI _PSD object (discussed in Chapter 5) informs the OS of its role and responsibility in coordinating P-state transitions. If all the logical processors in a CPU share the same P-state domain, it is not necessary for every logical processor to request a P-state change. OSPM policies have the ability to decide which of those logical processors should ultimately own the decision. Similarly, if all the logical processors in a CPU have their own unique P-state domain, the operating system must monitor each of them individually and make unique, logical, processor-specific requests.

Hardware provides several mechanisms OSPM can use to get feedback on policy decisions. The IA32_PERF_STATUS MSR provides a measure of the current frequency ratio, which conveys the specific processor P-state a core is utilizing. This mechanism is frequently used by software outside the OS to show the current operating conditions. Another mechanism for measuring frequency is the IA32_APERF and IA32_MPERF MSRs. IA32_APERF increments at the real frequency of a logical processor whereas IA32_MPERF increments at the CPU base frequency of the CPU. Measured over time, the ratio of IA32_APERF/IA32_MPERF allows OSPM to calculate the average frequency of a logical processor. Both IA32_APERF and IA32_MPERF can be reset by the operating system by writing them to zero, allowing OSPM to clear history from one observation window to the next. Some operating systems do not reset these registers, so IA32_APERF and IA32_MPERF can be used by other software outside the OS. Tables 6-4, 6-5, and 6-6 describe these mechanisms in greater detail.

www.dbooks.org

Table 6-4. *IA32_APERF MSR (0xE8)*

Bits	Width	Field	Description
63:0	64 bits	Actual performance	Always running measure of time. Increments when processor is active at the current operating frequency of the part.

Table 6-5. *IA32_MPERF MSR (0xE7)*

Bits	Width	Field	Description
63:0	63:0	Measured performance	Always running measure of time. Increments when processor is active at the CPU base frequency of the part.

Table 6-6. *IA32_PERF_STATUS MSR (0x198)*

Bits	Width	Field	Description
7:0	8 bits	Reserved	Unused
14:8	7 bits	Core ratio	Frequency ratio of the core. This is multiplied by the base clock (typically 100 MHz) to get core frequency.
63:15	49 bits	Reserved	Unused

Collaborative Interface

A controversial topic that has been debated for years between hardware, firmware, and software engineers is, "Who is the ideal entity to control P-states?" Each of these entities has some unique information that provides insight to making an optimal P-state selection. For example, the OS has unique information about processes and threads, their priority, and dependencies between them. Hardware has unique insight into power, temperature, leakage, and resource scalability. Hardware can continuously monitor behavior and detect changes faster than an OS. Management firmware may understand a critical need to migrate a virtual machine (VM) off a server or have unique knowledge about response time requirements. A collaborative interface that allows control entities to specify requirements and hints, rather than discrete P-states, is a step toward a mutual decision.

Some of the latest CPUs contain support for hardware-controlled performance states (HWP), an interface for collaborative decision-making between hardware and software. HWP gives software the ability to supply a target frequency range to operate within along with performance guidance hints. This allows software to use its unique information to provide guidance and for hardware to optimize the selection of P-states within those software provided constraints. If software has no unique information to provide, hardware has the ability to autonomously select P-states.

There are a number of new capabilities this interface provides for control, feedback, and notifications that don't exist in the legacy interface. For example, this interface has the capability to indicate a change to guaranteed frequency initiated by an external power or thermal control policy. Tables 6-7 and 6-8 highlight the most important interface capabilities—the primary ones used by OSPM to cooperatively manage P-states. HWP MSRs are documented in detail in the Intel Architecture Software Developer Manuals.

Table 6-7. *IA32_HWP_CAPABILITIES MSR (0x771)*

Field	Description
Highest performance	Value for the maximum non-guaranteed performance level (aka Turbo).
Guaranteed performance	Current value for the guaranteed performance level. Can change dynamically as a result of internal or external constraints (e.g., thermal or power limits).
Most efficient performance	Value of the most efficient performance level (aka P-state with the lowest voltage).
Lowest performance	Value of the lowest performance level.
Reserved	Unused.

Table 6-8. *IA32_HWP_REQUEST MSR (0x774)*

Field	Description
Minimum performance	Minimum performance allowed in P-state selection.
Maximum performance	Maximum performance allowed in P-state selection.
Desired performance	A performance hint to hardware within the performance range defined by IA32_HWP_CAPABILTIES (see Table 6-7). If set to zero, hardware makes this decision autonomously.
Energy performance preference	A performance hint to hardware that influences the rate of performance increase/decrease and the result of hardware optimizations.
Reserved	Optional or unused.

www.dbooks.org

Firmware Control

The previous sections detailed some of the challenges software faces in the selection of P-states, such as how to strike a balance between low power and high performance that will meet most users' needs. Another challenge software faces is ensuring that new power management features work correctly on older versions of software.

This is a commonly faced challenge as new hardware is frequently coupled with old software. For example, a server utilizing hardware released in 2014 may be running software released in 2009. *Forklift upgrades*, where software and hardware are simultaneously upgraded, are uncommon due to cost, complexity, and integration risk. In many datacenters, hardware platforms are upgraded or replaced several times under the same software stack. This mismatch between the introduction of new hardware and new software creates an environment where servers may not be fully enabled or optimized for the latest power management features. This is very different from client products that can rely on new hardware being coupled with new software as well as some product-specific device drivers for power and thermal management. Figure 6-3 illustrates the differences in energy efficiency between different Linux kernels without changing the server, applications, or workload. Between 2008 and 2012, idle power decreased by 35 W and active power at a given throughput level decreased by up to 15 W.

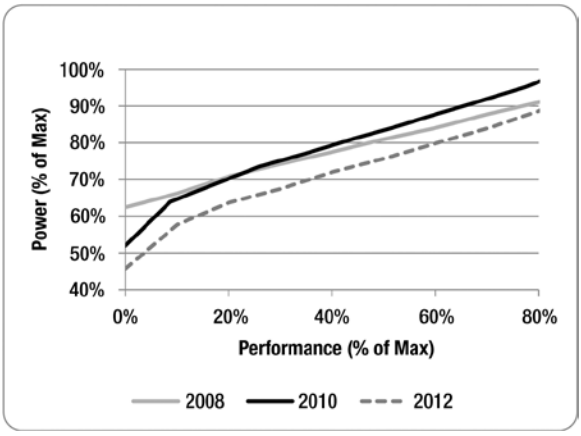


Figure 6-3. *Power impact of different Linux kernels*

Another challenge in a datacenter is enforcing a consistent set of power and performance tradeoffs across systems running different operating systems. Each OSPM policy makes its own unique choices about power state selection leading to unique behaviors. One may be biased toward maximizing performance while another may be biased toward minimizing power. There are cases where it is desirable to have a single OS-agnostic power management policy that can be applied across many different software environments.

It is not practical for an administrator to continually upgrade software to enable the latest hardware feature support and optimizations. Similarly, it is not practical for an administrator to fine-tune one OS to behave like another. To address these issues, several

server manufacturers provide their own P-state control capability that is implemented in firmware. This allows an administrator to apply a uniform P-state control policy to an entire rack of servers that may be using a variety of different operating systems and versions. Similar to many OSPM implementations, these firmware policies allow administrators to specify a preference toward low power, high performance, or a balanced setting. These types of advanced configuration options are available as BIOS setup options, allowing administrators to select between firmware or software-controlled policies.

P-state Policy

P-state policy is responsible for adjusting the performance capability of the processor to meet current or expected demand. When logical processor utilization is low, demand can be met by running the processor at low frequency. As logical processor utilization increases, frequency must also increase to prevent the system from reaching full utilization.

Similar to C-states, P-state policy is based primarily on logical processor utilization. Individual policies may supplement utilization with additional information such as the current operating frequency, or the priority of processes running. Utilization calculation and time window considerations discussed for C-states are similar for P-states including the use of IA32_PERF_CTR1 and IA32_TSC. Similar to C-states, the OS makes individual decisions on behalf of each logical processor where the ACPI P-state domain indicates this is necessary.

Performance Capacity

There are two ways a server can increase throughput, or the number of instructions executed per second. The first is to utilize available resources. This is as simple as utilizing idle cycles on a processor whenever that processor is not fully utilized. The second is to increase the speed or rate at which a logical processor executes instructions. This is accomplished through increasing frequency.

There are many cases where processor utilization alone is not sufficient to understand the operating conditions of a processor. For example, if a processor is 50% utilized running at 1.0 GHz (its minimum frequency), that processor has significantly more performance headroom than if it was 50% utilized running at 3.0 GHz (its maximum frequency). Although processor utilization is the same in both cases, the operating conditions are very different.

An important concept to introduce that aids in the understanding of processor operating conditions and OSPM P-state policies is performance *capacity*. This metric is based on processor utilization, but it takes into account the impact of running at a higher or lower frequency to accurately capture performance headroom. The capacity metric represents the amount of the maximum guaranteed throughput capability the processor is currently using.

$$Capacity\,(\%) = Utilization\ at\ CPU\ Base\ Frequency * \frac{Current\ Frequency}{CPU\ Base\ Frequency}$$

www.dbooks.org

A capacity of 100% might represent a processor running at 100% utilization while at the CPU base frequency. This is the guaranteed throughput capability of the processor. With Turbo, which is a non-guaranteed frequency, it is possible for capacity to exceed 100%.

Figure 6-4 illustrates how OSPM uses frequency to achieve maximum throughput. Up to 40% of maximum throughput is achieved at the lowest frequency, simply by utilizing idle cycles. At the point at which logical processor utilization starts to approach 100%, OSPM P-state policy increases frequency to provide additional throughput. When the processor reaches 90% of maximum throughput, it is running at the highest frequency, and the last 10% of idle cycles are utilized to reach maximum throughput. This chart also serves as an illustration of a P-state policy designed for best energy efficiency.

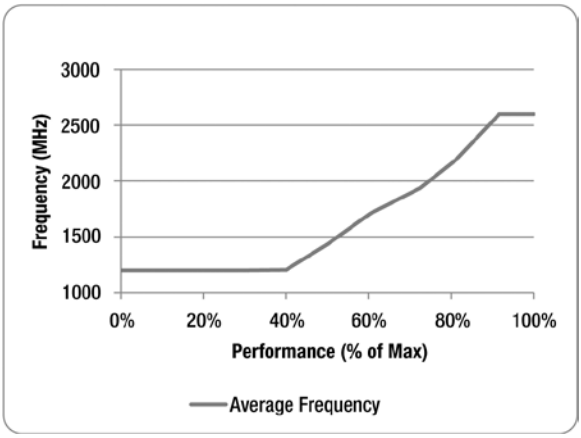


Figure 6-4. *Comparison of frequency across a full range of performance*

From a capacity standpoint, a 100% utilized processor running at 1.5 GHz and a 50% utilized processor running at 3.0 GHz are identical. Running at high utilization and low frequency provides lower power and higher response times whereas running at low utilization and high frequency provides higher power and lower response times. The performance impact of P-states is realized in a datacenter in terms of higher or lower response times.

Where there is sustained load on the system, a P-state policy should never impact maximum throughput. Use of P-states varies between operating systems, and operation at a given capacity can be accomplished in many ways. Whether to meet an increase in compute demand by running at higher utilization or by running at higher frequency is a key decision OSPM P-state policy needs to manage. Figure 6-5 shows a comparison between average logical processor capacity and average logical processor utilization.

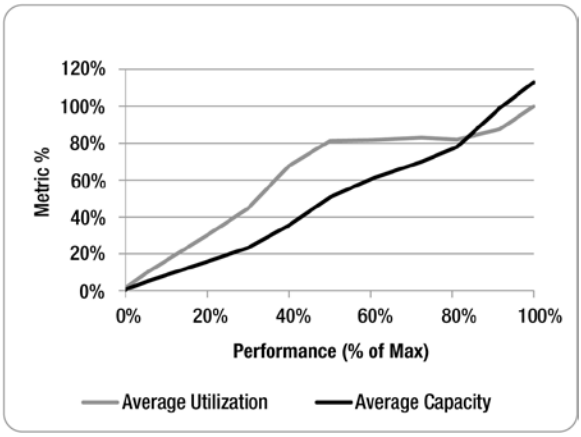


Figure 6-5. *Comparison between average utilization and average capacity*

■ **Note** Datacenter administrators typically describe server utilization in terms of logical processor utilization, or what is displayed by monitoring and profiling tools. Describing utilization without taking into account frequency is terribly misleading since a server running at 40% utilization may in fact only be running at 25% capacity.

In Figure 6-6, capacity exceeds 100% because the metric is based on the CPU base frequency, or the guaranteed frequency of the processor. As discussed earlier, Turbo mode is a non-guaranteed frequency, so the chart illustrates performance due to Turbo being above and beyond the guaranteed performance capacity of the processor. Capacity isn't a perfect metric either; it assumes exceptional frequency scaling (ratio of the percentage increase in performance to the percentage increase in frequency). In reality, frequency scaling varies from system to system since it is heavily dependent on the performance of the cache and memory subsystem and is influenced by unique workload and system characteristics.

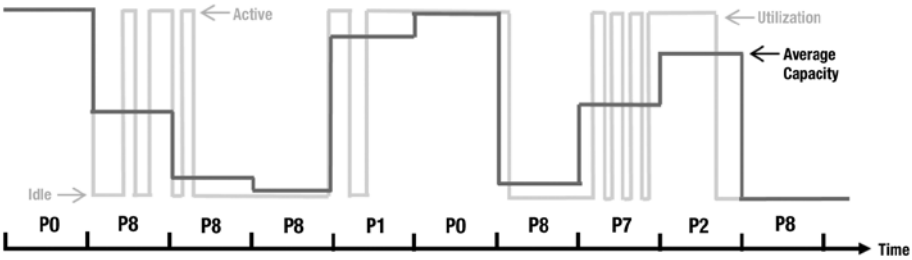


Figure 6-6. *Sample logical processor utilization with requested P-state over one second*

www.dbooks.org

Figure 6-6 shows sample capacity from a single logical processor and energy efficient use of P-states over one second of time. For simplicity, the example assumes nine P-states with the lowest frequency mode at P8, the CPU base frequency of the processor at P1, and Turbo at P0. During periods of light activity, P8 is frequently selected, even when the capacity is greater than 30%. Nearly half of the throughput capability of the processor can be satisfied by utilizing idle cycles at P8 alone.

There are cases outside of Turbo where the CPU will run logical processors at a higher frequency than the OS requests. The integration of memory controllers and I/O (PCIe) in the processor creates cases where logical processor utilization alone is insufficient to determine if frequency is limiting performance. For example, it is possible for network intensive workloads to drive line-rate traffic at very low processor utilization. Due to this low processor utilization, a typical OSPM policy will select a low frequency P-state. However, in this case, running logical processors at the highest frequency will increase throughput through decreasing latency. With OSPM lacking visibility into these types of bottlenecks, modern processors include special features to detect and handle these cases. The CPU will autonomously increase processor frequency in cases of high IO bandwidth to ensure maximum throughput is not impacted.

P-state Coordination

With P-states, as is the case for T-states and C-states, it is possible to change the way coordination happens between software and hardware. Specified via ACPI, power state coordination can use one of the following methods. (The vast majority of server operating systems utilize HW_ALL.)

- **HW_ALL:** The OS makes state transition requests for each logical processor and treats them as independent. The OS does not need to consider any effects on other logical processors. Hardware takes care of any coordination needed. Where there are two logical processors in the same core requesting P1, and one of the logical processors changes its request to P*n*, the core remains at P1.
- **SW_ANY:** The OS makes a state transition request for a single logical processor in the same domain and the effect is immediate and independent of previous requests. Where there are two logical processors in the same core requesting P1, and one of the logical processors requests P*n*, the core goes to P*n*.
- **SW_ALL:** The OS makes state transition requests for each logical processor in the same domain and treats them as dependent. For example, if there are sixteen logical processors in a single domain, the OS changes P-state for that domain by making the same request on each of the sixteen logical processors.

■ **Note** While P-state coordination is controlled by BIOS firmware, selecting a mode other than HW_ALL can result in undesirable behavior because most operating systems have not been enabled or optimized for other modes.

T-state Control

Operating systems also have the capability to control T-states, or throttling states, described in Chapter 2. Modern server processors all include the appropriate temperature sensors and hardware mechanisms to prevent the processor from overheating, so OS control of T-states is no longer necessary. Using T-states to improve energy efficiency is generally not done nor is it recommended. Unlike P-states, voltage is not scaled for T-states—using T-states results in substantial decreases in performance for only small decreases in power.

Global Power Policy

There are cases where the default C-state or P-state policy set by the OS may be out of sync with user performance requirements. Operating systems have two different mechanisms for tuning power management to address this. The most common is support for predefined power policies. Operating systems provide options for the administrator to set a global power management policy that will change the behavior and selection of C-states and P-states to be more biased toward low power, high performance, or a balanced approach. For example, the Windows Server control panel exposes options such as high performance, balanced, and power saver that change the way the internal OSPM algorithms select C-states and P-states. A second type of tuning is possible using advanced settings or options accessed using low-level tools or interfaces that allow administrators to customize their own policy.

As discussed in Chapter 2, there are a variety of other power management features outside of C-states and P-states that impact power and performance. Most of these power states across memory, caches, and processor interconnects are not explicitly controlled by OSPM. The visibility required to make power state decisions only exists in hardware. Although OSPM has no low-level control over these power features, hardware does expose a mechanism for the OS to specify an energy policy preference.

Energy policy preference, or OSPM bias toward low power or high performance, is controlled by the IA32_ENERGY_PERF_BIAS MSR described in Table 6-9. The register is used to define the desired balance between power and performance for those power management features that are not explicitly controlled by OSPM. In environments with strict response time requirements, it's preferable to use the minimum IA32_ENERGY_PERF_BIAS value rather than disabling features, since disabling power management features can increase power and temperature, ultimately impacting maximum performance. IA32_ENERGY_PERF_BIAS is dynamic so it can be changed at runtime as OSPM sees fit.

www.dbooks.org

Table 6-9. *IA32_ENERGY_PERF_BIAS MSR (0x1B0)*

Bits	Width	Field	Description
3:0	4 bits	Energy policy preference hint	Represents a sliding scale where the value 0 is maximum performance and the value 15 is minimum energy. A value of 7 roughly translates to a balanced policy.
63:15	49 bits	Reserved	Unused

Process Scheduling

The OS process scheduler doesn't directly interact with hardware power management features, but its scheduling policies can have a significant effect on energy efficiency. Many scheduler features that improve performance, such as starvation prevention (ensuring every process gets a chance to run), unique treatment of CPU and I/O bound processes, load balancing, and migration optimizations, have a positive impact on energy efficiency. However, there are many features where performance benefit has a negative impact. In addition, the answer to the question of whether or not various scheduler features improve energy efficiency can depend on the target microarchitecture. Some of the key factors that impact kernel scheduler energy efficiency include awareness of logical processor, core, and package topology and capabilities, timer tick frequency, and execution consolidation.

Topology and Capability Awareness

A key piece of information an OS scheduler needs in order to make energy-efficient decisions is to what degree multiple hardware threads or logical processors on the same core share resources. Where there is very little resource sharing between logical processors, a process scheduler can treat them as independent execution resources. However, where there is substantial resource sharing between logical processors, a process scheduler needs to understand how the throughput capability of an individual logical processor is impacted as additional logical processors sharing the same resources are concurrently utilized.

With a large number of servers running at low utilization, the process scheduler has a key decision to make in terms of how to utilize logical processors. One strategy is to utilize a single logical processor on every core before utilizing any of the simultaneous multi-threading (SMT) "sibling" logical processors. This has the benefit of giving logical processors dedicated access to otherwise shared core resources, keeping logical processor utilization low. Low logical processor utilization leads OSPM to use lower power P-states. Another strategy is to utilize both logical processors on a core before utilizing any of the additional cores. This limits coherency traffic and minimizes resource use, improving C-state residency. It may allow some of the cores to remain in long uninterrupted idle durations.

There is no single right answer for the best strategy to pursue since the optimal decision is microarchitecture specific. Your strategy depends on the amount of resources shared between logical processors; it is influenced by cache sizes and levels of the cache hierarchy; and it is subject to the different types of C-states available and the latency to transition in and out of them. To some extent, it even depends on unique workload characteristics. Generally speaking, the majority of server processors benefit from spreading software threads out over as many cores as possible. Typically, the scheduling decision that keeps core frequency lowest is the most energy efficient. When a scheduler utilizes all logical processors on the same core before scheduling on the next cores, resource contention increases utilization and leads to OSPM use of higher power P-states earlier. It takes a substantial increase in C-state residency to offset even a single step increase in P-states.

Another strategy from a process scheduling standpoint is to utilize all the logical processors on one CPU before utilizing logical processors on the other CPU. Conceptually, this makes a lot of sense as a CPU executing no instructions should be able to enter very low power states. In reality, benefits with this approach are mixed. In a multi-socket system with two or more CPUs, any single CPU cannot independently go to a deep package C-state while any other CPU in the system is active. There can be a significant amount of memory and I/O device activity on a CPU even when all its logical processors are idle due to the integration of the memory controller and PCIe.

Another challenge with utilizing only logical processors on one CPU is the impact to the non-uniform memory access (NUMA) locality. If remote memory references increase as a result of a scheduling decision, it can increase response times and decrease energy efficiency.

OSPM P-state policies that are optimized for performance may see an energy efficiency benefit to this approach as they use high-frequency P-states even at low utilization. In these environments, utilizing all the logical processors on one CPU before utilizing logical processors on the other CPU limits the number of CPUs running at high voltage.

Timer Tick Frequency

A periodic timer interrupt determines the frequency at which the operating system will perform necessary scheduling tasks. This periodic timer or timer tick plays a significant role in energy efficiency, both when the processor is active and when it's idle. When idle, logical processors must wake up to handle timer ticks, interrupting the coordination necessary to enter a deep package C-state. Modern operating systems suppress the timer tick or limit the number of timer ticks when the system is idle. For example, at idle, timer ticks may be received by a single logical processor that is responsible for forwarding the interrupt only to active logical processors. This improves the average idle duration for logical processors and results in improved package C-state residency. This action is commonly called tick skipping, dynamic ticks, or tickless idle.

When the processor is partially utilized, high-frequency timer ticks can lead to higher performance, but this comes at a power cost. A kernel with a 1000 Hz tick rate delivers a timer interrupt every 1 millisecond whereas a kernel with a 64 Hz tick rate delivers a timer tick every 15.6 milliseconds. Linux has the option to get rid of timer ticks even when busy.

www.dbooks.org

From a performance perspective, running with a higher frequency tick results in lower response time. The increased time accuracy from a higher frequency tick improves the resolution of timed events—it leads to more precise process preemption and it improves enforcement of priority and fairness policies. For example, a high-priority process that is reading data from a drive will have less latency between drive reads with a high-frequency timer tick. From a power perspective, systems running with a higher frequency timer spend more time handling timer interrupts. This increase in interrupt handling time drives logical processor utilization higher and splits long idle durations needed for deeper C-state entry into several shorter idle durations. Increased interrupt rates also increase the number of C-state transitions, accumulating more C-state exit latency and adding more transitional energy.

Execution Consolidation (Core Parking)

A process scheduler spreads software threads across as many logical processors as possible, avoiding scheduling multiple software threads on the same logical processor until there are no more free logical processors available. This maximizes use of parallel resources and minimizes process response time. There are some cases from a performance, power, or thermal perspective when it is preferable to do the opposite—scheduling multiple threads on a single logical processor to leave other logical processors idle. When execution is consolidated in this manner, the OS must also ensure the handling of device interrupts and other timed events is done only on the subset of active logical processors.

Energy Efficiency

Scheduling software threads to maximize utilization on a single logical processor while leaving other logical processors idle increases deep C-state residency and minimizes power state transitions. Execution consolidation has many different names in products and research such as core parking, core idling, and power-aware scheduling. Creating an imbalanced load in this manner keeps some number of logical processors in an uninterrupted idle state.

Although execution consolidation does increase deep C-state residency, it doesn't necessarily improve energy efficiency—whether or not this practice benefits energy efficiency is dependent on many factors. One key consideration is the amount of power that comes from the cores in comparison to the remaining uncore components, such as the last level cache, memory controllers, and processor interconnects. Execution consolidation is not favorable for CPUs that have a significant amount of power coming from uncore components. It takes only a single active logical processor to keep the uncore components in an active state. In these cases, the percentage increase in power from running a workload on two cores compared to one core is very small. Execution consolidation is more favorable where a CPU has non-core resources, such as mid-level caches, shared by small subsets of cores, but not all cores. The more that cores and resources shared by small subsets of cores account for total CPU power, the more favorable execution consolidation becomes.

Execution consolidation can increase response times as only one software thread can run at any given time on a logical processor. With execution consolidation, software threads that are ready to execute may need to wait to execute to avoid utilizing those

logical processors kept in an uninterrupted idle state. This practice typically results in active cores running a high utilization, also limiting the time a CPU can utilize package C-state states. Another consideration is the degree to which the software threads share data. Where there is substantial data sharing, particularly data that is frequently modified, execution consolidation can improve cache locality significantly. This characteristic is not clearly visible to a process scheduler, so dynamically identifying beneficial cases is difficult. Where cache benefits are realized, the reduction in execution time and decrease in interconnect and memory utilization can offset some or all of the cost of increased execution time. The same execution consolidation concept can extend to CPUs in multi-socket systems. However, it is common for the memory controller and I/O (PCIe) in the CPU to be active even if no software threads are running. This uncore activity consumes a significant amount of power. As a result, simply moving software threads off of a socket will not allow that socket to enter a low power idle state.

In measuring energy efficiency, a key consideration is how execution consolidation is used in conjunction with P-states. For most processors, actions taken by the process scheduler that cause increases in logical processor utilization typically lead to increased use of higher frequency P-states. In these cases, energy efficiency suffers because even very small increases in core voltage result in very large increases in power. In practice, it takes very specific or synthetic workloads on very specific processors to show an energy efficiency benefit from execution consolidation, but these cases are the minority.

Power Capping

A more universally beneficial application of execution consolidation is for *power capping*, that is, managing the system so it is always operating below a defined power limit. The primary mechanism for server power limiting is by placing hard limits on logical processor frequency and memory bandwidth. When these mechanisms are exhausted and the power limit is still not met, platform firmware can initiate logical processor idling to reduce power further. In cases where there is a failure in power or cooling infrastructure, it may take several different power limiting mechanisms to meet the power limit and avoid system shutdown.

ACPI includes an optional processor aggregator device that provides an interface and control point for firmware to idle logical processors. When additional power limiting is necessary, firmware requests a specific number of logical processors using the ACPI _PUR method, and the OS satisfies this request.

There is no energy efficiency benefit to power limiting when the practice is characterized on a single server. However, in cases where a rack of servers are operating under a rack-level power limit, datacenter management software can limit power of less critical servers in order to give additional power budget to more critical ones.

Single-Threaded Performance

Another application of execution consolidation is to improve single-threaded performance. As discussed in Chapter 2, Turbo frequency increases with the number of idle cores in the CPU. Restricting the key software thread along with device interrupt handling, timed events, and any other background activity all to a single core ensures the active core is always running at the maximum Turbo frequency.

www.dbooks.org

Memory Management

Similar to the concept of execution consolidation, if a server is using less than its full memory capacity, it is possible to consolidate memory references to a subset of memory. This allows remaining memory to enter uninterrupted low-power states. There has been significant investigation in this area, but only a limited number of scenarios realize a benefit.

Memory consolidation requires the OS to understand the physical memory topology. Similar to CPU topology, this information is conveyed to the OS through ACPI. The Memory Power State Table (MPST) describes physical memory in terms of memory power nodes, or specific address ranges that are power managed as a single entity. There can be several address ranges within a single node because these ranges may not necessarily be contiguous. In addition, MPST describes the power states supported by hardware including power consumption and exit latencies. Similar to OSPM for C-states and P-states, this level of information is critical for the memory manager, so it has the right inputs to determine power saving and performance impact.

Unlike C-states and P-states, the OS is not required to initiate power state transitions for memory nodes. Hardware autonomously transitions power nodes between appropriate power states based on their activity level. Memory nodes are typically defined at the channel level instead of a rank or DIMM level. The reason for this is because the lowest power states for memory, such as self-refresh, are applied at the channel level. Figure 6-7 shows the relationship between physical memory and memory power nodes. A memory power node could include from one to three DIMMs.

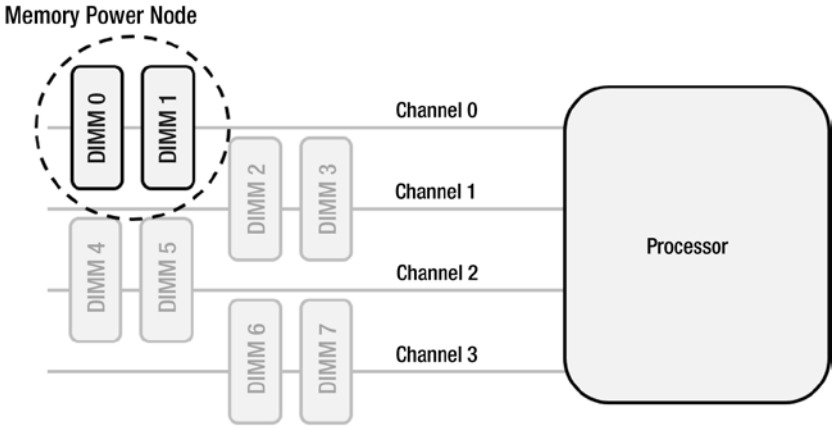


Figure 6-7. *Relationship between physical memory and memory power nodes*

Equipped with the physical memory topology, latency, and power information, an OS memory manager can make power-aware decisions about memory allocation. First, physical memory can be allocated so that DIMMs are used to capacity on one channel before DIMMs are used from remaining channels. Next, the OS memory manager can periodically relocate physical memory. This is necessary as a large amount of unused physical memory is uncommon and memory becomes fragmented over time. Relocation puts frequently referenced memory into one subset of power nodes, and infrequently referenced or unallocated memory into another subset of power nodes. Although this practice doesn't ensure that one subset of nodes is always idle, it does allow for significant residency improvements in the deepest power states.

As is the case with other power management features described in this chapter, memory consolidation needs to balance the need for low power with the need for high performance. The default settings for a server are to interleave memory across channels and across ranks of memory. This increases the bandwidth capability of the system by spreading large regions of allocated memory evenly across channels, ranks, and DIMMs. In order to enable power-aware memory management, this interleaving needs to be disabled by BIOS firmware when the memory controllers are initialized. From a memory bandwidth perspective, many datacenter workloads use only a fraction of the bandwidth capability, so no performance impact is realized. For other workloads with higher memory bandwidth requirements, this is an unacceptable trade off. It is possible for an operating system to maintain some level of memory interleaving to accommodate bandwidth demands, but the overhead of managing this interleaving in software would eliminate any of the potential benefits.

Another challenge facing memory reference consolidation is the complexity of relocating memory. Relocating memory adds overhead in terms of additional processor time and additional memory traffic, both of which reduce the time spent in low-power states. Workloads that change their memory reference characteristics over time or memory pages that are repurposed over time make it difficult to predict what pages will be hot or cold. Finally, not all regions of memory can be relocated, such as reserved or non-paged memory.

The potential benefit of memory consolidation is being reduced over time as datacenters transition from DDR3 1.5 V to DDR3L 1.35 V to DDR4 1.2 V. Not only is active memory power lower, but the power differences between shallow memory power states applied at a rank level and deep memory power states applied at a channel level are decreasing. The complexity of power-aware memory management leads to limited applicability. It requires very specific workloads and system configurations to show significant energy efficiency benefits.

Device Drivers

In addition to processor drivers that implement C-state and P-state control policies, some I/O device drivers are also responsible for implementing device power management and control policies. Other devices autonomously manage power and do not define software interfaces for power state discovery and control. Most servers can't tolerate the latency of deep D-states in a production environment. In cases where the latency impact is negligible, some devices monitor their own activity and utilize shallow states. In other cases, many D-states features are unused or disabled on servers.

www.dbooks.org

PCIe, SATA, and USB

As discussed earlier in the chapter, operating systems use PCI-SIG defined standardized interfaces for discovering PCIe D-state capabilities, for putting devices into a D-state, and for querying the power status of devices. The OS PCI driver discovers the supported capabilities from PCI configuration space when it enumerates the PCI devices. D-states for PCIe, SATA, and USB devices are managed through similar native hardware interfaces. ACPI is used to augment or supplement these capabilities' for example, to handle devices that lack native hardware interfaces, such as end-point devices on I2C (standard bus for attaching low-speed peripherals). Similar to C-states and P-states, software coordinates with hardware to initiate D-state transitions.

Software-initiated D-states require a greater level of coordination than do processor power states because they involve device, host, bus, and class drivers. Device drivers are responsible for saving and restoring device context before and after devices are put into low-power states. In addition, the OS has the additional responsibility of ensuring devices have no pending transactions before initiating entry into a D-state. Additionally, the OS must ensure that all devices on a bus are in a low-power state before putting a bus in a low-power state. Due to the latency of device power management, server operating systems typically initiate D-states based on entry into a higher level system state, such as S3. Prior to entering S3, the OS is required to put each device into a D3 state. The OS maintains specific mappings between S-states and different D-states bus and device components must go into prior to entering a target S-state.

Graphics

Modern server processors may also include integrated graphics processors. One of the more complex cases of I/O device driver power management is for graphics because, in addition to the responsibility of managing D-states, the graphics driver must also manage graphics processor P-states. Similar to a processor driver, the graphics driver's P-state request is also based on events that measure the current level of activity. Device drivers use a combination of demand, latency, and frame per second (fps) requirements in determining the appropriate P-state. The graphics driver updates the PCU with information in addition to the required core and ring performance to keep graphics running effectively.

With integrated graphics, processor cores and graphics share the same package power and thermal budget. This is unlike most servers, which feature a discrete graphics controller. With integrated graphics, it's not possible for both graphics and processing cores to be active in the highest frequency state at the same time. Two different device drivers making requests for higher performance and power states on the same CPU without coordination between them can cause issues where either the processor cores or graphics aren't getting their requested performance. When the power budget cannot satisfy the requests of both drivers, the PCU makes decisions based on its own internal knowledge to best balance the power budget.

Graphics devices also utilize C-states, but these are controlled autonomously in hardware. Graphics hardware detects when resources are idle and handles C-state entry including context save and restore. C-states are immediately exited whenever graphics hardware is accessed by the device driver.

Virtualization

A virtual machine monitor (VMM) has all the same responsibilities for power management that a native operating system does. The VMM, or host, is responsible for controlling C-states, P-states, and global power policy. In addition to these mechanisms, VMMs enable several additional capabilities that improve energy efficiency such as server consolidation or new approaches to power management enabled by virtual machine (VM) migration.

Power State Control

VMM management of C-states, P-states, and global power policy is not a responsibility shared with guest VMs. These features are host-controlled. If guest VMs were allowed to access power state control capabilities, it would create a number of policy conflicts between concurrently running VMs. Some VMs, with no knowledge of other VMs, would require high performance, biasing state selection toward shallow idle states and high-power active states. Some VMs would require low performance, biasing state selection toward deep idle states and low-power active states, and other VMs would request everything else in-between.

Allowing the host to control power management and limiting or eliminating the role of guest VMs is common practice across the various models for virtualization, such as the hypervisor model used in ESXi, the host-based model used by Hyper-V and the kernel-based virtual machine, or KVM, and the hybrid model used by Xen. Each of these VMMs boots VMs using virtual BIOS. The virtual BIOS exposed to guest VMs has a different set of capabilities than the physical server's own BIOS firmware. To facilitate host-controlled power management, the virtual BIOS does not expose power management features to the guest VM. This prevents guest VMs from unnecessarily loading drivers and control policy for C-states and P-states—all guest VMs need to do is execute a halt. This eliminates additional overhead introduced by guests making state requests that would be ultimately ignored by the host.

There are some cases where guests are *enlightened*, or aware they are running in a virtualized environment. These guests may be given certain control capabilities or may have an awareness of power management features that is not typical of guest VMs. This enlightenment allows for some power management optimizations by the host. For example, different C-states or S-states may be exposed to guests, allowing individual VM's power state selections to act as feedback for the host. These requests allow a host power management policy to consider individual VM requests along with its own policy to make optimal system-level decisions.

Idle Considerations

Virtualized environments present some unique challenges to getting to low idle power. Even an operating system doing nothing generates significant activity when there are many different guest VM operating systems loaded. Similar to native environments, or environments that run a single operating system, some minimal amount of activity exists at idle, such as periodic timer interrupts or network heartbeats. These periodic events continually wake up logical processors and can prevent the system from entering deep

www.dbooks.org

package C-states. Even a minimal amount of activity becomes significant when it is multiplied by a large number of VMs on a single physical system.

Similar to native operating systems, many VMM have optimized idle scenarios in an attempt to align periodic events and reduce the number of times logical processors are woken up. For example, timer ticks for different VMs (especially those sharing a common frequency), can be aligned to happen at the same time. This causes periodic event handling for multiple VMs to happen in parallel, reducing the number of power state transitions and increasing average idle residency.

Figure 6-8 compares VMM and guest idle scenarios across several years of software improvements. In the figure, the base system idle power is approximately 100 watts. The 2009 software stack uses a combination of guest operating systems released at that time and earlier, with varying degrees of idle activity optimization. The VMM in this case is doing little to no periodic activity alignment. The 2014 software stack uses a combination of guest operating systems released at that time and earlier and the VMM aligns periodic activity. This comparison illustrates the importance of software components in achieving low idle power in a virtualized environment.

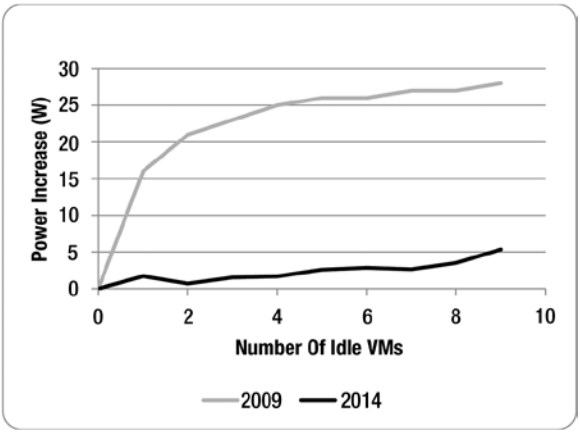


Figure 6-8. *Idle power impact due to software in a virtualized environment*

Active Considerations

Utilization characteristics in a virtualized environment are typically different than a native environment. Guest VMs are limited in the number of logical processors, the amount of memory, and the amount of network bandwidth they can utilize. These limits vary based on instance type and VM size. Restricting applications to different subsets of system resources leads to asymmetric resource utilization. Resource utilization in a native environment is typically very similar across processors and sockets because applications have the ability to utilize all available system resources. In a virtualized environment, it's not uncommon to see some logical processors running at sustained 100% utilization, while others are idle or at low utilization. These significant differences in utilization characteristics affect the OSPM policy's ability to make P-state decisions that accommodate a wide variety of VM performance requirements.

Figures 6-9 and 6-10 illustrate typical logical processor utilization at a fixed throughput rate. In the example of a native environment, the system experiences very few changes in utilization characteristics and many of the logical processors have similar utilization levels. In the example of a virtualized environment, the system experiences frequent changes in utilization characteristics with great variation in utilization across logical processors. These qualities lead to a challenging set of decisions for OSPM. In order to meet the performance requirements of a wide variety of VMs, virtualized environments are typically much less aggressive in their use of power management.

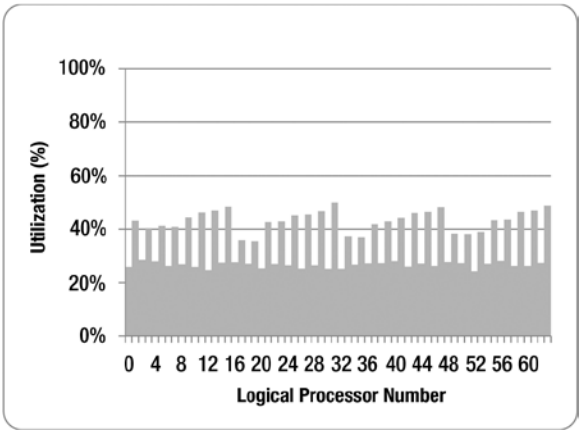


Figure 6-9. *Example of 30% throughput in a native environment*

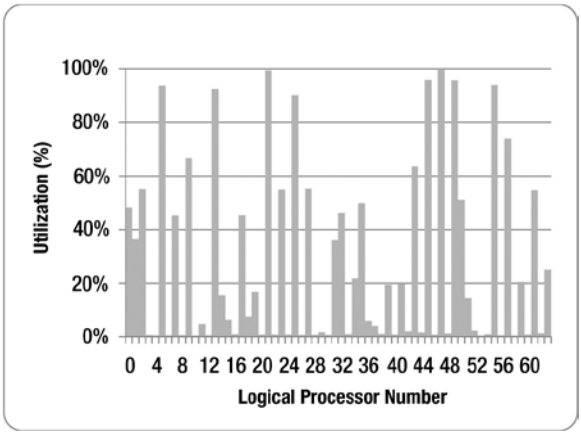


Figure 6-10. *Example of 30% throughput in a virtualized environment*

www.dbooks.org

Another consideration from an energy efficiency standpoint is the overhead of additional software layers present in a virtualized environment. This overhead can increase execution or response time and decrease throughput. Either of these impacts increases power as transactions or computational tasks take longer to complete. The impact of virtualization overhead increases as more virtual servers are consolidated onto the same physical server. CPU bound workloads tend to experience less impact from overhead, typically less than 10%, since there is little to no kernel time. I/O-bound workloads that execute lots of system calls, retire a lot of privileged instructions, and generate a lot of interrupts typically see greater than 15% overhead.

As a result, software or hardware enhancements that limit or eliminate the overhead of a VMM provide significant improvements in energy efficiency. Virtualization technologies such as EPTs (extended page tables) or VT-d (virtualization technology for direct device assignment) are typically thought of as performance optimizations, but running workloads on systems with and without these features enabled demonstrate their capability as a power optimization.

Consolidation

The majority of native servers, or servers that run a single operating system, support only one primary application. Limiting the number of applications has several benefits in this scenario. It improves performance, it avoids resource conflicts, it improves the ability to monitor applications, and it encapsulates problems. In many cases, running a single application on a server also leads to low CPU utilization.

The single-best method for improving energy efficiency of a server is to increase utilization. The energy cost of a server transaction is inversely proportional to utilization. For example, the average energy cost of a server transaction is up to five times lower at 60% capacity than it is at 10% capacity. Server consolidation is one of the primary mechanisms for increasing utilization. It allows a number of existing servers, such as servers running at low utilization or servers running with low-performance processors, to be replaced by a single higher performance server. The operating systems and applications from existing servers are consolidated and deployed as guest VMs running under a VMM on the higher performance server. Even though multiple applications are running on the same physical server, these applications continue to gain many of the same isolation benefits realized with a 1:1 mapping between applications and physical servers.

The benefits of consolidation from an energy efficiency perspective are immense. The primary benefit is the power savings realized by decreasing the number of physical servers required to support the same set of applications. Most virtualized environments have greater than a 10:1 consolidation ratio (the number of virtual servers running on a single physical server), so the opportunity to save power is tremendous. After server consolidation, many of the older and higher powered servers can be powered off and removed completely. The next benefit of consolidation is the ability to increase the average utilization of servers. Physical servers running a VMM supporting multiple applications typically have much higher utilization than native servers supporting a single application.

■ **Note** Consolidation is often limited by non-CPU constraints such as the amount of DRAM in the platform or performance requirements defined in service level agreements.

Servers running legacy software, or older applications and operating systems, are frequent targets for consolidation. Older operating systems may not have optimal support for the latest hardware power management features or may have some power management features by default. For example, environments such as Windows Server 2003 or Linux distributions using pre-2.6.5 kernels do not enable the most beneficial power management features out of the box. In this context, consolidation provides additional benefit by enabling legacy software environments to use the latest OSPM policies. When guests with legacy software are run under a modern VMM, many of the latest software power management benefits are realized.

VM Migration

A server running at 25% of maximum performance is energy proportional if it consumes no more than 25% of maximum power. Technologies such as C-states, P-states, and memory CKE (Clock Enable) are key ingredients in the pursuit of energy proportionality; however, servers still have a long way to go to meet this goal.

Migration allows VMMs to move a guest VM from one physical server to another without service interruption or loss of execution context. There are many reasons to move a VM from one physical system to another. The most common reasons are to perform maintenance on a system or to vacate a system that is experiencing some type of performance or functional issue. Datacenter management software also uses VM migration to load balance virtual servers across physical servers. Using migration and consolidation together, some servers can remain at high utilization where the energy cost of transactions is minimized. The remaining servers can be suspended (S3) or powered off (S5). Figure 6-11 illustrates energy proportionality through migration and consolidation. With migration and consolidation represented by this example, energy efficiency can be improved by more than 25%.

www.dbooks.org

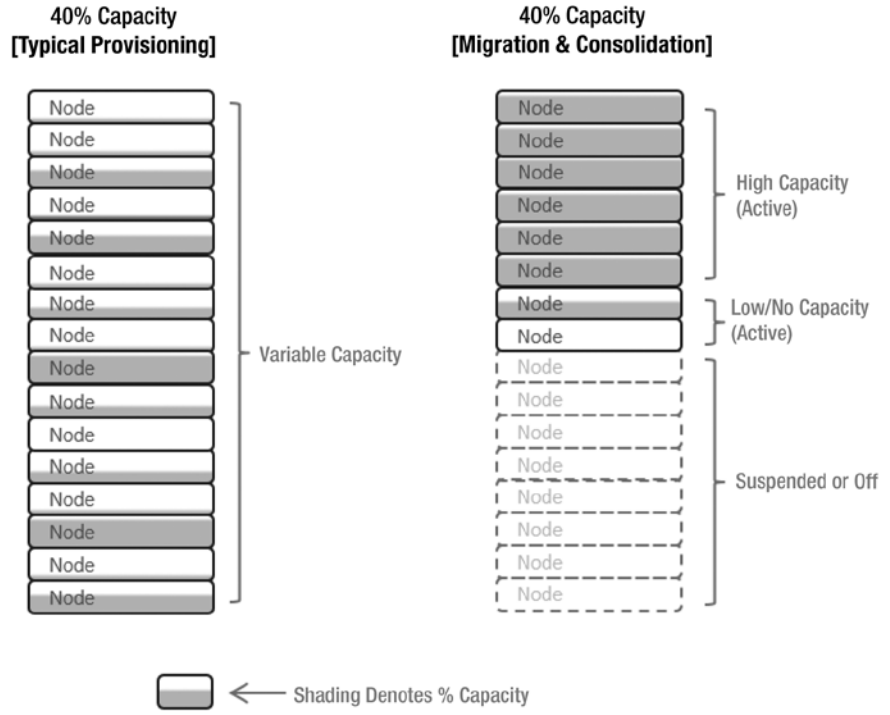


Figure 6-11. *Example of efficiency through migration and consolidation*

Inactive servers can be awakened by wake-on-LAN packets or by standard out-of-band management interfaces such as the Intelligent Platform Management Interface (IPMI). It is also common to keep some number of servers idle, but not suspended. When new VMs are created, these instances can be provisioned on unutilized but active servers, hiding the latency of initializing suspended servers. These practices allow for a more dynamic resource pool where energy efficiency is maximized and fluctuations in throughput are accommodated without waiting for servers to enter and exit S-states.

Migration is gaining adoption to improve utilization and energy efficiency, but the practice is not widespread today. There are a number of reasons why adoption is inhibited. First, VM migration is very expensive from a latency perspective. It can take several minutes to migrate a VM depending on the workload, application, and active memory footprint. During the migration, the average response time of transactional workloads increases sharply. For computational workloads, the maximum throughput decreases. Applications utilizing local storage in either case present additional challenges. Many datacenters lack the capability to accurately predict current and future demand. Transitions in and out of S-states exhibit high latency. Exiting S3 can take several seconds to resume where there is significant execution context that needs to be restored. Hardware and software innovations will continue to accelerate VM migration time and decrease S-state transition time making this a more viable energy efficiency strategy moving forward.

Comparison of Operating Environments

All OSes use the same mechanisms for power management discovery and control, however the strategy and use of these interfaces is very different. Each OS makes a unique set of decisions on how to measure activity, how frequently to change power states, and what is acceptable performance impact. Each OS determines whether these decisions are self-contained or influenced by outside services, management, and orchestration software. Each OS determines the amount of customization and tuning it will allow and how different combinations of OS decisions map to higher level global power policies.

The following sections provide details on unique characteristics of each of the most broadly deployed server operating systems. The section outlines unique traits and behavior of OSPM default settings including a look at the balance between power and performance. Major feature and policy changes of each OS are outlined to compare specific OS versions with different power management features and capabilities described in this chapter.

Microsoft Windows Server (including Hyper-V)

Windows Server OSPM is optimized for best energy efficiency by default. Applications running in this environment consume less energy, minimizing cost. Significant increases in response times are realized as a result of the focus on lower power. As is the case across all operating systems, OSPM policies do not typically impact maximum performance. Power management features that impact performance are not used when there is sustained high CPU utilization.

The Windows Server P-state policy is capacity driven, increasing frequency when utilization passes a predefined threshold. The threshold to increase frequency is high, ensuring that use of higher frequency only happens when the server is no longer able to accommodate demand at the current frequency. The policy is more aggressive in decreasing frequency when utilization decreases than it is in increasing frequency when utilization increases, leading to lower power. The Windows Server P-state policy does an excellent job of utilizing the full range of ACPI exposed P-states. Utilization is observed over tens of milliseconds and the OSPM P-state policy has the ability to maintain past history of utilization.

The Windows Server C-state policy is simple, examining utilization over a window of tens of milliseconds. The C-state policy makes a single target state decision based on the last observation window that is used throughout the current observation window. The simplicity of the policy has the advantage of being non-intrusive, adding no latency to the C-state entry path in software. With hardware demotion mechanisms filtering out non-optimal C-state decisions, the solution provides outstanding energy efficiency.

Windows Server supports user-configurable global power policies including *power saver*, *balanced*, and *high performance*. Each of the power policies represents a combination of individual OSPM parameters that control C-state and P-state policy decisions. The behavior of each of these parameters and features is highly configurable. For example, it is easy to set a P-state floor or ceiling, modify thresholds for frequency increase or decrease, or change the duration of the observation window used to measure utilization. An example of this is in high-performance mode where P-states below the advertised frequency of the CPU are not used.

www.dbooks.org

It is typically much easier to tune OSPM policies to decrease response time than it is to tune policies to decrease power, making the Windows Server default behavior a good starting point for administrators looking to fine-tune OSPM to meet specific performance requirements. The power policies also change IA32_ENERGY_PERF_BIAS, adjusting hardware power management features to meet the desired balance between power and performance specified by the OS. Several of the key Windows Server tuning mechanisms are described in detail in Chapter 7.

Windows Server also includes advanced power features that can provide additional benefits in some special cases. These advanced features can provide power saving with a subset of specific workloads running on specific systems. Where feature benefits don't apply to most workloads and systems, the advanced features are typically disabled by default. The *core parking* feature in Windows Server consolidates execution to improve energy efficiency. It dynamically adjusts the number of logical processors used for running software threads, allowing some logical processors to enter deep, uninterrupted idle states. The Windows Server utility distribution feature can be coupled with core parking. Utility distribution monitors activity that cannot easily be relocated from one logical processor to another, such as software threads or interrupts affinitized to a specific logical processor. Windows Server uses this information to improve core parking decisions and to improve prediction of future demand. The *memory cooling* feature consolidates memory references to a limited set of memory power domains, saving power for systems with large memory capacity where significant portions of memory are not frequently utilized.

OSPM is largely the same between Windows Server and Hyper-V, but the specific parameters and tuning values that define power saver, balanced, and high-performance power policies vary slightly between the two. Hyper-V includes some minor changes that trade off some of the power savings for improved response times.

Table 6-10 identifies major power management features and improvements added to Windows Server over time.

Table 6-10. *Historical Changes in Microsoft Server Operating Environments*

Version	Released	Changes
Windows Server 2003	2003	Added support for C-states.
		Added support for P-states.
		Added support for T-states.
		Added global power policy (user selectable).
		Default policy is high-performance (results in P-states not used by default).
		No independent logical processor control for P-states and C-states, only a single processor supported.

(*continued*)

Table 6-10. (*continued*)

Version	Released	Changes
Windows Server 2003 R2	2005	No significant changes to Windows Server 2003.
Windows Server 2008 (and Hyper-V role)	2008	Default processor performance policy is balanced (results in P-states used by default). Added power policy for ASPM.
Windows Server 2008 R2 (and Hyper-V role)	2009	New processor power management policies with significant energy efficiency improvements. Added timer tick coalescing. Added intelligent timer tick distribution (tick skipping). Added core parking. Added power metering and budgeting. Added remote power management and group policy. Hyper-V now built in. Hyper-V live migration.
Windows Server 2012	2012	Added logical processor idling. Added memory cooling. Hyper-V supports more VMs and more processors per VM (4 to 64).
Windows Server 2012 R2	2013	No significant changes to Windows Server 2012.

Linux Distributions (including KVM)

Linux OSPM is optimized for low latency. Applications running in this environment have improved performance, minimizing transaction response times. A significant increase in power is realized as a result of the focus on lower latency. Similar to other operating systems, the OSPM policies do not impact maximum throughput. Power management features that may impact performance are not used under sustained high CPU utilization.

Linux supports P-states through the CPUfreq infrastructure which provides interfaces for low-level control drivers and high-level control policies, called *governors*. Governors can be dynamically changed, but the default for most server distributions is *ondemand*. Ondemand is capacity driven, and selects the highest available frequency when utilization is above a predefined threshold. When utilization falls below the threshold, the next lowest frequency is used. Under variable loads, frequency is increased more aggressively than it is decreased, leading to low latency. When a server is partially utilized, ondemand tends to use a limited range of ACPI-exposed P-states, with the majority of requests being for P*n* or P0. This is due to the P0 being the only target state

www.dbooks.org

used whenever utilization exceeds the threshold. Ondemand observes utilization of tens of milliseconds and does not consider past history.

For environments without response time requirements that can tolerate additional latency, the *conservative* governor is an alternative to ondemand. This governor provides a more balanced use of the full range of P-states. In comparison to ondemand, the conservative governor results in higher response times, but with lower power. The *performance* governor can be used to permanently run at the highest frequency, and the *powersave* governor can be used to permanently run at the lowest frequency.

Intel also provides its own native P-state driver called `intel_pstate`. This driver is optimized for low response times and minimizes latency and the software overhead of P-state selection as the governor and scaling driver are combined. The driver is enhanced to understand the specific capabilities of each processor, which allows for improved use of Turbo. The `intel_pstate` driver can be described as a native driver, meaning it uses CPU interfaces to determine a richer set of information about power management capabilities instead of using ACPI. Native drivers are resilient to any issues the BIOS may introduce with incorrect ACPI objects.

Similar to P-states, Linux supports C-states through the CPUidle infrastructure, which provides the same separation between low-level control drivers and high-level control policies. The default governor for most server distributions is *menu*. OSPM policy uses a number of different metrics to make an optimal C-state decision. These include previous C-state residencies, expected idle duration, and the exit latency of target C-states. A target C-state is selected for every idle entry on every logical processor rather than determining a single target C-state for all logical processors over some time window. The advantage of this approach is that poor decisions are less frequent; the disadvantage of this approach is the cost of additional software overhead. The advantages typically outweigh the disadvantages, providing superior energy efficiency. Intel also provides its own low-level `intel_idle` driver that is enhanced to understand specific capabilities of each processor. `intel_idle` is able to expose more hardware C-states than an ACPI BIOS can expose to `acpi_idle`. For example, on some servers, `intel_idle` is able to export a C1 state with lower latency than C1E. Unlike `intel_pstate`, it does not replace existing higher-level policy.

Linux does not include global power policies that automatically change governors, their tuning, and platform-level controls such as IA32_ENERGY_PERF_BIAS, with a single setting. Rather, Linux relies on individually selecting and tuning C-state governors and P-state governors along with utilities such as *cpupower* to manage IA32_ENERGY_PERF_BIAS settings and options for execution consolidation. Linux has extensive capabilities to fine-tune individual power management parameters such as setting a minimum frequency or restricting use of a specific C-state. These are discussed in greater detail in Chapter 7.

Linux has several advanced power features that can provide additional benefits in certain environments. These features may work well with a specific workload and specific microarchitecture, but not well in others. As a result, some of these features are disabled by default. The scheduler supports core parking through cgroups and the CPU hotplug infrastructure. These features can be used to consolidate execution to a specific subset of logical processors. However, it must be coupled with additional management software

to enable dynamic execution consolidation. Energy efficiency benefits of execution consolidation are more common when coupled with a low-latency P-state policy, such as ondemand. Energy efficiency benefits are less common when this technique is coupled with a balanced or low-power P-state policy.

■ **Note** The kernel-based virtual machine (KVM) inherits key power management functionality from Linux, so there are very few differences in power management capabilities or policies between a native and virtualized environment using KVM. Xen does not inherit key power management functionality in the same way. Power management features added to the Linux kernel have to be re-implemented or ported to Xen, so many of the features and explanations in this section do not directly apply to Xen in the same manner.

Table 6-11 identifies major power management features and improvements added to the Linux kernel over time. In some cases, key power management features have been back-ported to add the support to existing Linux distributions. Rather than cover every distribution, along with the kernel major and minor version numbers, the Linux reference introduces the kernel version in which a capability was first introduced.

Table 6-11. *Historical Changes in Linux Server Operating Environments*

Version	Released	Changes
Kernel 2.4.22	2003	ACPI built-in
		P-states disabled by default
Kernel 2.6.5	2004	Added CPUfreq subsystem
		Added ondemand and other governors
		P-states enabled by default (using ondemand governor)
Kernel 2.6.18	2006	Added support for deep C-states (I/O port)
Kernel 2.6.19	2006	Added MWAIT support for deep C-states
		Added APERF/MPERF feedback used for P-states
Kernel 2.6.23	2007	ACPI OSI (Linux) disabled by default
Kernel 2.6.24	2008	Added tickless idle (CONFIG_NO_HZ)
		Added CPUidle subsystem
Kernel 2.6.30	2009	Added USB HID autosuspend
Kernel 2.6.32	2009	Added `acpi_pad` (processor aggregator device) for power limiting

(*continued*)

www.dbooks.org

Table 6-11. (*continued*)

Version	Released	Changes
Kernel 2.6.35	2010	Added `intel_idle` (Intel CPUidle C-state driver)
Kernel 2.6.36	2010	Added support for deep C-states in CPU offline Added USB mass storage autosuspend
Kernel 3.1	2011	Added kernel support for IA32_ENERGY_PERF_BIAS (kernel sets this to 6 if found it is 0 at boot-time)
Kernel 3.5	2012	Removed `sched_mc_power_savings` scheduler tunable (early dynamic execution consolidation implementation)
Kernel 3.9	2013	Added `intel_pstate` (native Intel P-state driver) Added idle injection driver
Kernel 3.10	2013	Added full tickless (CONFIG_NO_HZ_FULL) allowing Linux to be built with no clock ticks, either when idle or busy
Kernel 3.11	2013	Added native RAPL driver for in-band power limiting

VMWare ESX and ESXi

In contrast to the approach taken by other operating systems, a key focus for ESXi-based environments is managing power at a cluster level, between a larger numbers of servers. Distributed Power Management (DPM) consolidates active VMs to a subset of servers, running that subset of active servers at higher utilization, while placing unutilized servers in a standby or off mode. DPM can be configured to run VMs on the smallest subset of servers possible to achieve energy proportionality.

OSPM policies for C-states and P-states are covered by Host Power Management (HPM). With multiple VMs competing for system resources, ESX Host Power Management is optimized for low latency. Applications running in this environment have improved performance, minimizing transaction response times. Applications that run in this environment have lower response times, but that comes at the cost of higher power. Similar to other operating systems, power management features that may impact performance are not used under sustained high CPU utilization.

ESXi supports user-configurable power policies including *high performance*, *balanced*, *low power*, and *custom*. These policies include both a combination of power management parameters and associated tuning values as well as static enabling and disabling of features. For example, in high-performance mode, P-states and deep C-states are completely disabled. In balanced and low-power modes, both P-states and deep S-states are enabled. Low-power mode enables all power management features and includes more aggressive use of the lower power and higher latency states. Custom policy allows administrators to specify power management parameter values such as limiting C-states based on their exit latency or changing the observation window used to measure utilization.

Table 6-12 identifies major power management features and improvements added to ESXi over time.

Table 6-12. *Historical Changes in VMWare ESX/ESXi Operating Environments*

Version	Released	Changes
VMware ESX 3.5	2007	Added Distributed Power Management (dynamic migration of VMs, shutdown and restart of servers to manage power).
VMware ESX 4.0	2009	Added support for P-states (disabled by default).
VMware ESX 4.1	2010	Added global power policy (user selectable). Default policy is high-performance (results in P-states not used by default).
VMware ESXi 5.0	2011	Added Host Power Management. Default policy is balanced (enables P-states by default, but no C-states).
VMware ESXi 5.1	2012	No significant changes.
VMware ESXi 5.5	2013	Added C-state support to balanced policy.

Summary

Operating systems play a key role in selecting both idle and active power states for the server. This is a difficult balancing act because the OS decisions heavily impact both performance and power. In addition to power state selection, OS process scheduling, I/O, interrupt handling, and memory management decisions also have a significant impact on power. Virtualized environments include many of the same capabilities as native environments. They also enable new usage models, such as migration and consolidation, which can provide substantial improvements in energy efficiency.

The list of historical changes across operating systems at the end of the chapter serves as a reference to highlight both the power management limitations of legacy operating systems as well as the latest power management enhancements in modern operating systems.

www.dbooks.org

CHAPTER 7

Monitoring

Optimizing a server for power, performance, or cost can be achieved with minimal effort. This process begins with monitoring the behavior of a system to understand how it is used and what opportunities exist for improvement. Sensor measurements such as voltage, current, energy, power, and temperature identify individual components that contribute the most to overall server power. Latency, bandwidth, and throughput events illustrate the energy cost of delivering additional performance. Monitoring of power state transitions, clock interrupts, and device interrupts allows users to build a deeper understanding of the distribution of work on a server. This chapter introduces various monitoring capabilities and how they work. It discusses various events and metrics, how these are collected, and what a user can learn from these. Chapter 8 continues with a description of how these events and statistics can be used to guide optimization decisions.

System and subcomponent monitoring helps users to improve component selection and future system design. Monitoring aids in software optimization and in identifying issues and opportunities to improve resource usage. Control decisions in management software utilize monitoring, adapting infrastructure to meet changing conditions. For example, management software can monitor processor utilization and memory bandwidth to guide VM migration decisions. Or a CPU can use thermal sensors to identify when to throttle processors down to a lower frequency.

Monitoring features are spread across several subcomponents in the platform. Processors have programmable performance monitoring units in the cores and uncore, baseboards are equipped with power and thermal sensors monitored by management controllers, and operating systems monitor individual application processor, memory, and I/O use. Comparing monitoring data from different subcomponents allows users to build a complete picture of how power and performance affect energy efficiency.

Hardware Monitoring

There are a variety of mechanisms for extracting monitored events or statistics from the CPU. Some of these high-level mechanisms are summarized in Table 7-1. Although each of these mechanisms includes some unique features and capabilities, it is not uncommon for certain events to be tracked through multiple mechanisms.

Table 7-1. *Types of CPU Hardware Monitoring*

Mechanism	Description
Performance monitoring counters (fixed counters)	These counters continually track a single fixed statistic. In many cases they are *free-running*, meaning they continuously count and cannot be stopped or cleared. Many critical hardware statistics are maintained in fixed counters.
Performance monitoring counters (programmable)	Hardware performance monitoring is frequently used by software developers for characterizing and optimizing their code. The majority of these counters can be configured to count a wide range of events. Performance monitoring counters exist inside the cores as well as in the uncore.
Status snapshots	Registers provide a snapshot of some system state. Software can read this state at a given point in time to understand the characteristics of the system. The most common example of a snapshot is temperature status registers.

There are a variety of different mechanisms for accessing CPU power statistics. However, the majority of these capabilities require kernel-level permission (administrator in Windows, root in Linux). If you do not have these privileges, these statistics are only available if the system administrator allows common users to access them. These statistics are also commonly not available inside of virtual machines, because the statistics are intended for the system as a whole.

Fixed Counters

A number of fixed counters are available in the system for tracking various statistics. These counters cannot be stopped or cleared (except through CPU cold boot or warm reset). These counters are very useful where the ability to access a common statistic is needed by multiple users at the same time. The downside of fixed counters is that they are restricted from some of the more powerful monitoring techniques.

Core Performance Monitors

Cores on Intel CPUs have used a standard performance monitoring infrastructure for many generations. Dedicated configurable counters (typically four) exist for each hardware thread. A configuration register exists for each counter that allows users to select a specific *event* to count. In addition to the four configurable counters, three fixed counters exist on the core. All of these are implemented as MSRs and are considered a part of the core performance monitoring architecture. The core performance monitors are covered in detail in the *Intel Software Developers Manual* (SDM) and therefore will not be covered in detail here. See the following resources for more information:

www.dbooks.org

- www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf.
- Core performance monitoring events are included as part of the SDM by processor family since these events can change from generation to generation. Core performance monitoring events for each processor are also described in public files (tsv and json) available at the following link: https://download.01.org/perfmon/.

For monitoring power management, the three fixed counter MSRs (IA32_FIXED_CTR[0,1,2]) in the core can be very powerful. Each of these counters can be configured with the IA32_FIXED_CTR_CTRL MSR to either monitor a specific thread that is executing, or, in the case of processors with SMT, monitor all the threads on the core. The first fixed counter tracks instructions retired, which is a count of every instruction executed by the processor. The second counter tracks *unhalted cycles*, or cycles when the processor is actively executing instructions. The second counter always counts at the current operating frequency of the processor. The third counter tracks unhalted cycles similar to the second, except it always counts at the base frequency of the processor. Software can use the IA32_FIXED_CTR_CTRL MSR to configure fixed counters to monitor either a specific thread or all threads that share a core. This configuration is done by writing a specific field in the IA32_FIXED_CTR_CTRL MSR called *AnyThread*. The same MSR interface exposes options to track either user time or kernel time, or both.

Many software tools already exist for monitoring both fixed and programmable core performance monitors (a selection of these are discussed later in the chapter). The simplest way to use a counter is with *time-based sampling*, using the following steps:

1. Clear the counter (to avoid early overflow).
2. Configure the desired event in the configuration register and enable the counter.
3. Wait some amount of time (while applications execute).
4. Read the counter again.
5. Repeat steps 1–4.

More advanced techniques are also possible, such as *event-based sampling* (EBS). In EBS, instead of collecting samples over a fixed amount of time, counters are automatically stopped when one of the counters hits a desired value. At this point, an interrupt is generated that informs monitoring software to collect additional information about system and software state. EBS is not frequently used for monitoring power management and therefore will not be discussed here.

Uncore Performance Monitors

Unlike core performance monitoring, the performance monitoring architecture in the uncore is not standardized across all generations and product lines. A common architecture is used on Xeon E5/E7 products starting in the Sandy Bridge generation. Very few uncore performance monitoring capabilities have been productized on other Intel products, so this chapter will focus on those available in E5/E7.

Uncore performance monitoring introduced in Sandy Bridge is quite similar to the capabilities that exist in the core. Counters are distributed throughout the different blocks in the uncore, providing the ability to collect a large number of statistics simultaneously. Although the bulk of the power-related statistics exist in the counters in the PCU, there are power-relevant events in other blocks too.

Product-specific details of the uncore performance monitoring capabilities and registers are published whenever a product is launched. E5/E7 uncore performance monitoring documentation is included for the latest generation of products at the following links:

- www.intel.com/content/dam/www/public/us/en/documents/design-guides/xeon-e5-2600-uncore-guide.pdf
- www-ssl.intel.com/content/dam/www/public/us/en/documents/manuals/xeon-e5-2600-v2-uncore-manual.pdf
- www.intel.com/content/www/us/en/processors/xeon/xeon-e5-v3-uncore-performance-monitoring.html

Global Freeze/Unfreeze

A challenge with uncore performance monitoring is coordination between the large number of monitoring units spread across various uncore blocks. Simply reading all the counters across those blocks can take tens of microseconds. This latency can both perturb workload execution (particularly if sampling is done over very short time windows) and risk collecting statistics about the monitoring software as opposed to the workload of interest. As a result, the uncore performance monitoring architecture includes a global "freeze" and "unfreeze" capability. This capability attempts to start and stop all counters across various uncore blocks at the same time. Although the synchronization is not perfect, any delay in the freeze of uncore performance monitoring (also known as *skid*) is typically well below a microsecond.

Edge Detection and Average Time in State

Many monitoring events specify a condition that is counted for every cycle in which that condition is true. For example, if there is an event that monitors the time when a core is in a target C-state, that counter will increment every cycle when the core is in the target C-state. In addition to measuring time in state, it is also useful to be able to monitor the number of transitions in and out of a state. For example, you might want to be able to count the number of times a target C-state was entered. In order to avoid plumbing separate events for both time and transitions, *Edge Detect* hardware is used, which can transform any event that counts time (or cycles) into an event that counts transitions.

A common monitoring technique is to use one counter to monitor cycles and a separate counter to monitor edges. By using both these events, a user can calculate the average time in a state:

$$Average\,Time\,in\,State = \frac{(Total\,Time\,in\,State)}{(\#\,of\,Transitions)} = \frac{Event\,X}{Edges(Event\,X)}$$

www.dbooks.org

Standard Events and Occupancy Events

Many events simply increment by one in a given cycle. If, for example, you are monitoring the number of reads to DRAM, a *standard event* would increment by one for each read.

In performance monitoring, when doing queuing and latency analysis, it can be useful to measure the occupancy of different queues. *Occupancy events* can increment by one or more in each cycle. Occupancy events can also be used with a *threshold compare* to increment by one whenever a queue is at a configurable occupancy or larger.

For performance monitoring, occupancy events are commonly used in the following ways:

- Average latency in a queue

$$Average\ Latency\left(Time\ in\ Queue\right) = \frac{Accumulated\ Occupancy}{Queue\ Allocations}$$

- Average occupancy of the queue

$$Average\ Occupancy = \frac{Accumulated\ Occupancy}{Cycles}$$

$$Average\ Occupancy\ When\ Not\ Empty = \frac{Accumulated\ Occupancy}{Cycles\ with\ Occupancy > 0}$$

- Time at a given occupancy (or more)[1]

$$Percent\ Time\ with\ Occupancy \geq X = \frac{Occupancy\ \ with\ Threshold = X}{Cycles}$$

Occupancy events can also be applied to power management statistics despite "queues" physically not being a common part of power management. For example, one may want to understand the amount of time when all cores are simultaneously in a core C6 state. By looking at individual core residencies, it is impossible to tell how the different core activity lines up in time. By using an occupancy event for a core C6 state with a threshold set to the number of cores in the system, you can count the amount of time when all cores are simultaneously idle.

Status Snapshots

Snapshots provide an instantaneous view of system characteristics. These are particularly useful for information that does not change very often (like temperature). One drawback of status snapshots is that monitoring software can accidentally collect information about itself. For example, if processor frequency is being measured by monitoring software at a high sampling rate, it may cause frequency to increase higher than it normally would without monitoring software present.

[1]Multiple threshold events can be used at the same time across multiple counters to build histograms.

Counter Access and Counter Constraints

Reading or writing from counters and snapshots typically takes a moderate amount of time. It can take tens of clock cycles to access core programmable counters and hundreds of clock cycles to access uncore and other counters. By collecting a large amount of information from throughout, the system can perturb workloads. The amount of statistics collected can have a large impact on the size of sampling that can be performed. Typically, sampling faster than about every 10 ms can lead to significant workload perturbations.

Counters can be implemented in the processor in different ways. Large hardware counters can be quite expensive (area, power) and are not always desirable—particularly in area-constrained blocks or those that are replicated many times. It is also difficult or impossible to provide instantaneous information about certain types of information, for example, in counters for monitoring energy consumption. Statistics about energy are very expensive to monitor accurately through regular synchronous logic. As a result, some fixed counters are updated periodically with this information rather than instantaneously.

Events and Metrics

Monitoring mechanisms described in the preceding section support a wide variety of different measurable events such as core temperature, uncore P-state transitions, or cache lines transferred over a CPU interconnect. Events typically measure a precise low-level occurrence, behavior, or time in state. Events by themselves often do not provide meaningful insight into system or subcomponent behavior. As a result, it is common to use one or more low-level events to calculate a higher-level power or performance metric. For example, a common metric for memory performance is bandwidth. This is calculated based on a number of low-level events such as CAS commands, DDR frequency, and the measurement duration in DDR clock cycles.

Events and metrics include many different types of statistics. Some examples are time, temperature, energy, voltage, frequency, latency, and bandwidth. Specific events and metrics exist across various subcomponents providing insight into runtime characteristics of the system.

Time (RDTSC)

Most software environments provide mechanisms to access information about time in the system. Generally these APIs are sufficient for collecting information about elapsed time when collecting power and performance monitoring statistics.

There are many different ways to measure time in the system. x86 includes an instruction for getting time—the read time stamp counter (RDTSC), which is available both to user space applications and the kernel. Since it is possible for instructions to execute out of order, RDTSCP (a serializing version of RDTSC) is also provided. This version ensures all preceding instructions have completed before reading the time stamp counter. RDTSC is synchronized both across all the threads on a socket, and across threads in a multi-socket system. The RDTSC instruction returns a cycle count that increments at the rate of the base frequency of the processor. Although there is a centralized clock on each CPU, the time stamp counter is also maintained in the core, making it very fast to access. Unlike some performance monitoring counters, RDTSC counts through all power

www.dbooks.org

management states. Several other timers exist on the system, but the time stamp counter is preferred because many of them take much longer to access.

Most software tools will provide APIs to measure elapsed time, so making direct use of the RDTSC instruction is generally not necessary. Although many of the statistics and formulas in this chapter that rely on elapsed time will use delta TSC, this can be replaced with whatever time measurement is provided by a given software environment.

Basic Performance

Two basic performance metrics that are key to understanding energy efficiency are CPI and path length. *CPI*, or cycles per instruction, measures the average number of CPU cycles it takes to retire a single instruction. This is the inverse of the common performance metric IPC, or instructions per cycle. Low CPI occurs when work is computationally simple, where there is a lack of complex operations, and data references frequently hit in core caches. High CPI occurs when work is computationally complex, high-latency operations are frequent, and many data references need to be satisfied by memory. Software with low CPI maximizes the time various subcomponents can spend in low power idle states. It is more energy efficient than high CPI.

Path length measures the average number of instructions it takes to complete a single unit of work. This unit of work is commonly represented by the throughput metric for a workload of interest. For example, a unit of work might be a single HTTP transaction on a web server, a write to a database, a portion of a complex scientific computation, or a single drive read in an I/O testing tool. Beyond the obvious performance advantages, software with low path length also maximizes the time spent in low power idle states since fewer steps are necessary to complete a unit of work.

Performance can be improved either through completing work faster (decreasing CPI), or by completing work using fewer steps (decreasing path length). In the following formula, performance represents the time it takes to complete a unit of work. As a result, performance can generally be expressed by multiplying CPI by path length. The following formulas show how to calculate these basic performance metrics. These can be monitored per-logical processor, per-core, or per-application process, or they can be averaged to give a system level view.

$$Performance = CPI * Path\ Length$$

$$CPI = CPU_CLK_UNHALTED.THREAD / INST_RETIRED.ANY$$

$$Path\ Length = INST_RETIRED.ANY / Workload\ Throughput$$

Energy Use

Energy is monitored through both the socket and memory RAPL features (see Chapter 2 for more details). As part of RAPL, free running energy counters track the amount of joules that are consumed by the processor. All energy measured by RAPL is represented in the same format, called *energy units*. A single energy unit represents a fixed amount of microjoules of energy consumed. Measuring energy in larger granularity energy

units allows a high-resolution energy counter to cover a much broader range of values. However, the specific amount of microjoules an energy unit represents can change from processor generation to generation. The RAPL_POWER_UNIT MSR exposes the energy units that are used for RAPL on a given processor. Bits 12:8 show the energy units. As an example, Sandy Bridge presents a value of 0x10, which corresponds to ~15.3 microjoule increments (1/2^16).

■ **Note** Energy units can change from generation to generation. Haswell, for example, uses different units (~61 microjoules) than Sandy Bridge and Ivy Bridge (~15.3 microjoules).

The amount of energy and average power consumed over a time window can be measured with the following two equations:

$$Energy\left(Joules\right)=\frac{\Delta Energy\,Counter}{2\,\hat{}\left(MSR\ 0x606\right)\left[12:8\right]}$$

$$Average\,Power\left(Watts\right)=\frac{Energy\left(Joules\right)}{\Delta Time\left(Seconds\right)}$$

Table 7-2 lists the different types of energy statistics available on the processor.

Table 7-2. *Energy Statistics*

Statistic	Type	Description
Socket energy	Fixed free-running counter	Socket energy reports an estimate of the energy used by all the logic in the CPU package, including all power rails. Not all rails are actively measured, so the power reported here is strictly guidance.
DRAM energy	Fixed free-running counter	DRAM energy provides an estimate of the energy used by the DDR3/4 memory devices in the system. Note that support for DRAM RAPL does require some platform enabling, and not all systems support DRAM RAPL.
Core energy	Fixed free-running counter	The core energy counter was introduced on the Sandy Bridge generation, but was later dropped on the Haswell generation. It is expensive to provide accurate energy estimates for the core domain.

www.dbooks.org

■ **Note** Energy counters are only 32 bits today. On Sandy Bridge, for example, it was possible for the counters to roll over after a few minutes. Similar to a car odometer, when the energy counters overflow, they simply wrap around, starting over at zero. Software that makes use of these counters should detect and adjust for this overflow.

Temperature

Modern processors include numerous temperature sensors that are exposed to software. These sensors are generally only available to kernel-level software, because they typically exist in MSR register space. These sensors are sometimes called digital temperature sensors (DTS). Temperatures are usually quite accurate, particularly close to the throttling temperature point. As temperatures get colder, DTS accuracy tends to degrade.

Most server memory also includes temperature sensors on the DIMMs. This is always the case with RDIMMs, and almost always the case with ECC UDIMMs. These sensors are called thermal sensor on-die (TSOD). A single TSOD exists on the memory DIMM (typically in the middle of the DIMM) rather than having individual sensors in each device. Because DRAM devices can be quite long, it is not uncommon for there to be a large thermal gradient down the length of the DIMM. This is particularly the case where DIMMs are oriented in the same direction as the airflow in the platform. As air passes over devices as it moves down the DIMM, it heats up, causing the "last" device to be much warmer than the first. Platform designers take these gradients into account when designing their thermal solutions. Because the TSOD is in the middle of the devices, it is common for some devices to have higher temperatures and some to have lower temperatures.

Rather than exposing the actual temperature in degrees Celsius, temperature counters report the *margin to throttle* (or the delta between the maximum allowed temperature and the current temperature). The margin to throttle is measured at a core level through the IA32_THERM_STATUS MSR and at a package level through the IA32_PACKAGE_THERM_STATUS MSR. Package temperature reports the highest temperature across all sensors on the package. This includes any additional thermal sensors that may exist outside the cores. Traditionally, cores have been a hot spot on a server die, but this trend is starting to change on low-power server designs where a larger percentage of the package power budget is spent on I/O power. The maximum allowed temperature needed to calculate the actual temperature in degrees Celsius is measured by the TEMPERATURE_TARGET MSR.

The IA32_THERM_STATUS MSR is *thread-scoped*, meaning an individual thread can only access temperature information about itself. In some systems, multiple threads can share a single temperature sensor and therefore will always get the same result. In addition to reporting temperature, the IA32_THERM_STATUS MSR also reports log and status information about thermal throttling that may have occurred in the system. The IA32_PACKAGE_THERM_STATUS and TEMPERATURE_TARGET MSRs are *package-scoped*, meaning that all threads on a socket share the same register (and data). Table 7-3 lists the different types of temperature statistics available on the system.

Table 7-3. *Temperature Statistics*

Statistic	Type	Description
Core temperature	Status snapshot	Core temperatures are exposed through IA32_THERM_STATUS (MSR 0x19C) and TEMPERATURE_TARGET (MSR 0x1A2). $Maximum\ temperature(or\ DTSMAX) = TEMPERATURE_TARGET[23:16]$ $Target\ Offset = TEMPERATURE_TARGET[29:24]$ $Margin\ to\ Throttle = IA32_THERM_STATUS[22:16]$ $Temperature(C) = DTSMAX - Target\ Offset - Margin\ to\ Throttle$
Package temperature	Status snapshot	Package temperatures are exposed through $IA32_PACKAGE_THERM_STATUS(MSR\ X1B)$. $Margin\ to\ Throttle = IA32_PACKAGE_THERM_STATUS[22:16]$
Additional package temperature sensors	Status snapshot	Sensors do commonly exist outside the core. There is no standardized interface for accessing information about these thermal sensors, although their information is included in PACKAGE_ THERM_STATUS.
Memory DIMM temperature	Status snapshot	Memory DIMM temperature is also maintained in the package for systems that support TSOD DIMMs. Like with the other package sensors, there is no standardized register set for accessing temperature information. Although this information is not publically documented today, IPMI can be used at the platform level to monitor memory DIMM temperatures.

www.dbooks.org

Frequency and Voltage

Frequency is one of the most important statistics when it comes to power and performance. There are a wide range of mechanisms for monitoring the operating frequency in the system.

Two primary mechanisms are the thread-scoped IA32_PERF_STATUS and IA32_PERF_CTRL MSRs. The IA32_PERF_STATUS MSR holds the current frequency ratio of the thread that reads it. It also has a non-architectural field that provides guidance on the current operating voltage of the core. The voltage field does not exist on older generation server CPUs but is present in many of the current CPUs. The IA32_PERF_CTRL MSR is the same interface introduced in Chapter 6 that allows the operating system to request a frequency ratio for a given thread. Similar to the IA32_PERF_STATUS MSR, the IA32_PERF_CTRL MSR also has a field for requested voltage, however this is no longer used. Voltage is now autonomously controlled by the package, and writes to these bits have no impact on system behavior. Monitoring both of these registers together allows users to understand the relationship between requested frequency and granted frequency.

On E5/E7 processors starting with Haswell, the uncore has its own frequency and voltage (see Chapter 2 for details). These processors include an UNCORE_PERF_STATUS MSR that holds the current operating ratio of the uncore. This register is not architectural and generally is only exposed on systems that have dynamic control of the uncore ratio.

The IA32_APERF and IA32_MPERF MSRs can be used to measure average frequency over a user-defined time window. These free-running counters are sometimes also called ACNT and MCNT. Technically ACNT and MCNT have no architectural definition. Instead, only the ratio of the two is defined. ACNT counts at the frequency at which the thread is running whenever the thread is active. MCNT counts at the base frequency (P1) of the CPU whenever the thread is active. Neither of these MSRs count when the thread is halted in a thread C1 or deeper C-state.

Recent versions of Windows have started clearing both ACNT and MCNT in the kernel, making it unusable by other software tools. Older versions of Linux (before 2.6.29) also had this behavior. Linux no longer clears these MSRs at runtime so that other software tools (such as turbostat, which is included with the kernel) can make use of the MSRs as well.

Similar to ACNT and MCNT, the fixed counters can also be used to monitor the average frequency of either a core or a specific thread when it is active. This methodology automatically filters out time when a core is asleep. The IA32_FIXED_COUNTER1 MSR increments at the rate of the current frequency whenever a thread (or core) is not halted. The IA32_FIXED_COUNTER2 MSR increments at the rate of the base clock of the processor whenever a core (or thread) is not halted. As discussed in “Core Performance Monitors” earlier in this chapter, when the fixed counter AnyThread bits are set to 0, the counters measure average operating frequency of a specific thread while it is active. When the AnyThread bits are set to 1, the fixed counters measure average operating frequency of the core as a whole.

■ **Note** It is common for different threads in the system to report different average frequencies, even on processors that do not support per-core P-states. This is because the average is only taken while a core is active.

To measure average uncore frequency over a user-defined time window, clocks can be counted in the CPU caching agent using a programmable counter (see uncore monitoring links earlier in this chapter for programming information). The caching agent clocks are stopped in Package C6, so this needs to be taken into account when calculating the average frequency. Table 7-4 lists the different frequency and voltage statistics available on the system.

Table 7-4. *Frequency Statistics*

Statistic	Type	Description
Current core frequency	Status snapshot	Core frequency is exposed through IA32_PERF_STATUS (MSR 0x198). On Nehalem and Westmere generations: $Frequency(GHz) = \frac{4 * IA32_PERF_STATUS[15:8]}{30}$ On Sandy Bridge (and generations that follow it): $Frequency(GHz) = \frac{IA32_PERF_STATUS[15:8]}{10}$
Current core voltage	Status snapshot	Core voltage is exposed through IA32_PERF_STATUS (MSR 0x198) $Voltage(volts) = \frac{IA32_PERF_STATUS[47:32]}{2^{13}}$
Requested core frequency	Status snapshot	Requested frequency is exposed through IA32_PERF_CTRL (MSR 0x199). $Frequency(GHz) = \frac{IA32_PERF_STATUS[15:8]}{10}$
Current uncore frequency	Status snapshot	Uncore frequency is exposed through UNCORE_PERF_STATUS (MSR 0x621). $Frequency(GHz) = \frac{UNCORE_PERF_STATUS[6:0]}{10}$

(*continued*)

www.dbooks.org

Table 7-4. (*continued*)

Statistic	Type	Description
Average core frequency (APERF/MPERF)	Free-running counter	Average core frequency is exposed through IA32_APERF (MSR 0xE7) and IA32_MPERF (MSR 0xE8). $Average\ Frequency(GHz) = Base\ Frequency * \frac{\Delta APERF}{\Delta MPERF}$
Average core frequency (core performance monitoring)	Core performance monitor	Average core frequency is also exposed through IA32_FIXED_COUNTER1 and IA32_FIXED_COUNTER2. $Average\ Frequency(GHz) = Base\ Frequency * \frac{\Delta IA32_FIXED_COUNTER1}{\Delta IA32_FIXED_COUNTER1}$
Average uncore frequency	Uncore performance monitor	Average uncore frequency is exposed through a combination of programmable and free running counters. $Avg.\ Active\ Freq(GHz) = \frac{\Delta Caching\ Agent\ Clocks}{Sample\ Period(ns) - Package\ C6\ Residency(ns)}$
Frequency histograms	Status snapshot	Some processors support the ability to generate frequency histograms for either the core or uncore or both. Configuring these events can be challenging. The PCM tool (discussed later in this chapter) provides these capabilities on processors that support it.
Frequency transitions	Status snapshot	One can measure the number of frequency transitions occurring in the system. On Sandy Bridge and Ivy Bridge E5/E7, frequency transitions on a given socket can be measured by using Edge Detection on the FREQ_TRANS_CYCLES event. On Haswell E5/E7, an additional performance monitoring event was added that tracks the number of uncore frequency transitions. The FREQ_TRANS_CYCLES event still exists and now counts the total number of frequency transitions across all cores as well as the uncore. Note: these “cycles” events also provide a rough estimate of the number of cycles for performing frequency transitions, but in general, they do not provide highly accurate indications of how long software was prevented from executing code.

C-States

Tables 7-5 and 7-6 list the different C-state statistics available on the system.

Table 7-5. *Core C-State Hardware Statistics*

Statistic	Type	Description
Core C-state residency	Fixed free-running counters	Free-running counters have been added on many processor generations to track the C-state residency on each core. They count at the same rate as RDTSC (at the base frequency of the processor). • Core C1 is exposed through MSR 0x660*. • Core C3 is exposed through MSR 0x3FC. • Core C6 is exposed through MSR 0x3FD. These counters refer to the actual hardware C-state and not the ACPI C-state.
Core C0 residency	Core performance monitor	As discussed earlier in the chapter, IA32_FIXED_COUNTER2 (with AnyThread = 1) monitors cycles spent in Core C0 and counts at the base frequency of the processor.
Core C1 residency	Equation	On processors that do not include a Core C1 Residency MSR, it can be calculated through the following equation: $Deep\,Cstate\,Cycles = Core\,C6\,Cycles + Core\,C3\,Cycles$ $Core\,C1\,Cycles = TSC\,Cycles - Deep\,Cstate\,Cycles - Core\,C0\,Cycles$
Thread active (TC0)	Core performance monitor	The IA32_FIXED_COUNTER2 MSR (with AnyThread = 0) will monitor the amount of time that the current thread is in a TC0 state. It counts at the base frequency of the processor.

(*continued*)

www.dbooks.org

Table 7-5. (*continued*)

Statistic	Type	Description
Thread C1	Equation	Thread C1 is not easily measured (except on processors that support the Core C1 residency counter and do not support SMT). The following equation can be used: $Deep\,Cstate\,Cycles =$ $Core\,C6\,Cycles + Core\,C3\,Cycles$ $Thread\,C1\,Cycles =$ $TSC\,Cycles - Deep\,Cstate\,Cycles - Thread\,C0\,Cycles$
Core C-state occupancy	Uncore performance monitor	The uncore provides a performance monitor that tracks the number of cores in a particular state at a given point in time. This can be used to calculate the average number of active cores. In addition, users can use the thresholding logic in the uncore performance monitors to count cycles with a given number of cores active. See the Uncore Performance Monitoring guide for a given processor for details, or see the source code of PCM (more details later) for an example.

**Core C1 residency is currently only supported on Silvermont-based processors.*

Table 7-6. *Package C-State Hardware Statistics*

Statistic	Type	Description
Package C-state residency	Fixed free-running counters, performance monitor	Just like with core C-states, free running counters exist for measuring package C-state residency. • Package C2 is exposed through MSR 0x60D. • Package C3 is exposed through MSR 0x3F8. • Package C6 is exposed through MSR 0x3F9.
Package C-state transitions	Uncore performance monitor	The residency MSRs are great for monitoring residency, but it is not possible to measure transitions with them. As a result, the same events for measuring residency were added into the Uncore Performance Monitoring infrastructure on Ivy Bridge. Using these events for residency with edge detection provides the ability to monitor the number of transitions. You can calculate the average time spent in a package C-state (average idle periods across the entire node) with the following equation: $$Avg.Time\ in\ Package\ Cx = \frac{Package\ Cx\ \mathrm{Re}\ sidency}{Number\ of\ Cx\ Transitions}$$ You can also calculate the average time between package C-states: $$Avg.Time\ in\ Package\ Cx\ Entrances = \frac{TotalTime}{Number\ of\ Cx\ Transitions}$$

www.dbooks.org

Memory Power and Performance

Each memory controller channel on Xeon E5/E7 CPUs includes multiple uncore performance monitoring counters and one fixed counter that counts DCLK[2] cycles. On recent processor generations, the memory controller clock is gated in deep package states, and the performance monitors will stop counting in this state. Table 7-7 lists key memory power and performance statistics available on the system.

Table 7-7. *Memory Power and Performance Statistics*

Statistic	Type	Description
Bandwidth	Uncore performance monitor	CAS commands (see Chapter 2 for more details) refer to read and write commands issued to DRAM. By counting CAS commands, one can measure memory bandwidth. $Bandwith\ Utiliztion = \frac{CAS\ Commands * 4}{DCLK\ Cycles}$ $Bandwidth\left(\frac{GB}{s}\right) = Utiluization * DDR\ Frequency\left(GHz\right) * \frac{8 Bytes}{clock}$
Clock gated time	Uncore performance monitor	The fixed performance monitoring cycles counter in the memory controller will increment whenever the clocks are not gated. By measuring time between samples with RDTSC in conjunction with the fixed cycles counter, you can calculate time clock gated. $Percent\ Active = \frac{2 * \Delta DCLK\ Cycles}{\Delta RDTSC} * \frac{Base\ Frequency\left(CHz\right)}{DDR\ Frequency\left(CHz\right)}$
CKE	Uncore performance monitor	CKE is controlled per rank. The memory controller has an event that counts time spent when the CKE signal is high (and power is high). This can be subtracted from the total number of clocks to determine time spent with CKE low. CKE is always low when the memory controller clocks are gated. $Percent\ CKE\ Low = 1 - \frac{CKE\ High\ Cycles}{DCLK\ Cycles}$

(*continued*)

[2]DCLK = ½ the clock speed of the marked speed of the DDR memory.

Table 7-7. (*continued*)

Statistic	Type	Description
Self-refresh	Uncore performance monitor	Memory self-refresh is applied at the channel level. Whenever a channel is completely clock gated, it is also in the self-refresh state. Therefore, when one calculates actual time in self-refresh, the percent time spent in a clock-gated state should always be added to the total calculated by the performance monitor. Self-refresh is used primarily at idle. It is common to observe very little time spent in self-refresh and very little time spent clock gated, even when at low utilization. $Percent\ Self\ Refresh = 1 - Percent\ \ Active + \frac{Self\ Refresh\ Cycles}{DCLK\ Cycles}$

Memory bandwidth directly impacts memory power. Required memory bandwidth also can significantly impact purchasing decisions when building a system.

PCIe Power Management

Little visibility exists into PCIe power management from within the SoC.

QPI Power Management and Performance

A wide range of performance monitors are available for QPI. Tables 7-8 and 7-9 list key QPI power and performance statistics available on the system. The performance monitoring counters in the QPI block count at the clock rate of the logic in that block. The QPI link operates at very high frequencies measured in GT/s (giga transfers per second). Sandy Bridge, for example, operated at frequency up to 8 GT/s. In these designs, a single flit (an 80-bit unit of transfer) of data is transmitted in four transfers (or at a rate of 2 GHz in this example). Starting with Sandy Bridge, the clocks used for the QPI performance monitoring logic ran at half this frequency (or 1 GHz in this example), and the logic can process two flits per cycle.

www.dbooks.org

Table 7-8. *QPI Power Statistics*

Statistic	Type	Description
QPI frequency (GT/s)	Uncore performance monitor	You can measure the frequency of the QPI block for Sandy Bridge and Ivy Bridge with the following equation: $Avg.\frac{GT}{s} = 8 * \frac{\Delta QPI\ CLK\ Cycles}{Time(Nanoseconds)}$ You can measure the frequency of the QPI block for Haswell with the following equation: $Avg.\frac{GT}{s} = 4 * \frac{\Delta QPI\ CLK\ Cycles}{Time(Nanoseconds)}$ Note that this equation assumes an active system that is preventing any course-grained clock gating from occurring. Lower frequencies may be measured in systems that are nearly idle.
L0p time	Uncore performance monitor	Time spent in the L0p state can be monitored with the QPI L0p event. $\%QPI\ \ LOp = \frac{\Delta LOp\ Cycles}{\Delta QPI\ CLK\ Cycles}$ By using edge detect, one can also measure the number of L0p transitions. Note that L0p transitions block data transfers for only a very short amount of time.
L1 time	Uncore performance monitor	Time spent in the L1 state can be monitored with the QPI L1 event. On recent processor generations, L1 is used exclusively in the package C6 state, so there is little need to measure its residency separately. In fact, this QPI counter stops counting in many cases due to clock gating that occurs at the same time.

(*continued*)

Table 7-8. (*continued*)

Statistic	Type	Description
Clock gating	Uncore performance monitor	The entire QPI block can be clock gated on some processor generations when the link is in an L1 state. By measuring the QPI cycles, one can calculate the amount of time when the link is clock gated and in an L1 state. To do this, one must know the clock frequency of the QPI link first. $\%Clock\ Gated = 1 - \frac{\Delta LOp\,Cycles}{QPI\ Freq(GHz) * Time(Nanoseconds)}$

www.dbooks.org

Table 7-9. *QPI Performance Statistics*

Statistic	Type	Description
Bandwidth	Uncore Performance Monitor	QPI bandwidth is driven not only by the transmission of cache lines of data, but also by the transmission of additional information for maintaining coherency and QPI protocol. The QPI performance monitors can separate out the protocol overhead flits from the data flits. The "TxL_FLITS_G0" event (code 0x0) used with the following event masks provide this information. • Mask: 0000_0001b: Idle flits • Mask: 0000_0010b: Data flits • Mask: 0000_0100b: Protocol flits • Mask: 0000_0110b: Total used flits (other than Idle) Recall that when QPI is not in an L1 state, it is always transmitting flits. *Idle flits* are used when no actual information must be transmitted. As a result, a simple way to measure the link utilization is with the following equations. For Sandy Bridge and Ivy Bridge: $$Link\ Utilization(\%) = \frac{\Delta QPI\ FLits(Data + Protocol)}{2 * \Delta QPI\ Clocks}$$ For Haswell: $$Link\ Utilization(\%) = \frac{\Delta QPI\ FLits(Data + Protocol)}{\Delta QPI\ Clocks}$$ One can also subtract off cycles spent in QPI L1 in order to filter out long idle periods from the utilization metric, but this is generally not significant when monitoring active workloads. Data bandwidth (GB/s) is another interesting metric that gives an indication of expected QPI power. Each flit in QPI contains 8 Bytes of data. Therefore, data bandwidth can be calculated with the following: $$Data\ Bandwidth\left(\frac{GB}{S}\right) = \frac{8B * \Delta QPI\ Data\ Flits}{Time(Nanoseconds)}$$

Management Controller Monitoring

Management controllers in the system, including the Baseboard Management Controller (BMC) and the Management Engine (ME) in the PCH, provide broader platform-level monitoring functions. The BMC connects to various busses, sensors, and components in the system, allowing it to act as a centralized monitoring resource covering a large number of different system components. The BMC can also pair monitoring functions with threshold values to generate events, such as an indication that a component's temperature has exceeded a safe level.

Management controller monitoring complements the monitoring functions provided by the CPU and operating system. The BMC and ME provide access to many unique monitoring events that cannot be monitored elsewhere such as fan speed, power supplies, voltages, and general platform health. It also allows for monitoring while a system is booting, powered off, or unresponsive. As discussed in Chapter 5, management controller monitoring functions are accessed through IPMI.

Component Power Sensors

Great insight is gained by understanding individual component power consumption and how individual components add up to overall platform power. In an ideal monitoring solution, platform power (after PSU efficiency losses) would equal the sum of all the individually measured components. However, most servers can only measure CPU and memory power, leaving an incomplete picture.

■ **Note** A common question is "how does power break down in a system?" One engineering technique is to identify the few components that consume the most power and focus optimization efforts on those. The breakdown of power in a system changes significantly from server to server and many of the individual components in a system only represent a small percentage of overall power.

Additional sensors can be added to the baseboard to measure these missing components, reporting either power or current and voltage. During board design, the cost of adding these enhanced sensors necessary to calculate component power is relatively small, for example, adding VRs that expose readings over the SMBus interface or adding current sensors accessible over I2C. These additional sensors allow the Node Manager (NM) firmware to calculate energy for all components in the platform to identify where unaccounted power is coming from. For example, NM can expose individual energy measurements for PCH, LAN, fans, and the BMC. The current generation of Node Manager supports up to 32 additional monitoring devices so component energy can be monitored in fine-grained detail.

Similar to the use of the CPU energy events, NM energy events are coupled with timestamps so users can read the sensors periodically to calculate power over a desired time window.

www.dbooks.org

Synthetic Sensors

In order to expose more information about the platform, Node Manager adds new synthetic sensors for platform characteristics that cannot be easily measured. Node Manager 3.0 added the ability to report volumetric airflow and outlet temperature. Those values are calculated based on the server chassis characterization process and current readings from fans speed sensors, energy consumed by the platform, and inlet air temperature. External management software can create a heat map from information about the physical location of servers in the datacenter and the reported outlet temperatures. This information can be utilized to improve datacenter efficiency, for example, to dynamically manage cooling set points, identify hotspots, or optimize workload placement decisions.

Sensors and Events

Between the BMC and the ME, there are an extensive number of sensors, events, and metrics provided for monitoring. Table 7-10 provides an example of the leading events used to characterize server energy efficiency. Support for various sensors and the specific names of those sensors can vary by platform, so generic or typical names are used in the table. Many different types of sensors such as error indications, hard drive status, or fault and activity LED status are intentionally left out of the table to focus on those sensors most relevant for power.

Table 7-10. *Common Monitoring Events Accessible by IPMI*

Type	Description
PSU input power	Monitors power going into the power supply. This represents the total power of the node.
PSU output power	Monitors power going out of the power supply. Useful for understanding the efficiency of the power supply.
PSU current	Monitors PSU current. Useful for observing conditions such as over-current and verifying that the PSU is working within the design specification.
PSU voltage	Monitors PSU voltage. Useful for observing conditions such as under-voltage and verifying that the PSU is working within the design specification.
Voltage	Monitors voltage across various power rails. This is typically paired with some alert to ensure voltage is not higher than the expected threshold.

(*continued*)

Table 7-10. (*continued*)

Type	Description
CPU energy	Monitors energy by individual CPU. It is useful to monitor energy sensors over time in order to calculate power. One CPU consuming significantly more power than others warrants investigation. There may be an opportunity to improve energy efficiency by optimizing software.
Memory energy	Monitors energy by subset of DIMMs sharing a common VR. Useful for assessing the impact of component selection.
LAN energy	Monitors LAN energy by individual interface.
Fan energy	Monitors fan energy by individual fan. Useful in conjunction with Fan Tachometer and component temperatures for assessing the efficiency of fan speed control.
PCH energy	Monitors PCH energy including the ME.
BMC energy	Monitors BMC energy.
Chassis inlet temperature	Monitors temperature at the front panel where colder air is coming in.
Outlet temperature	Monitors temperature at the rear panel where hotter air is going out. Combined with inlet and component temperature, this is useful for understanding heat removal.
Riser inlet temperature	Monitors temperature at the given riser board location.
Riser outlet temperature	Monitors temperature at the given riser board location.
Board temperature	Monitors temperature at the baseboard.
PCH temperature	Monitors temperature at the PCH including the ME.
PSU temperature	Monitors temperature at the power supply.
CPU temperature	Monitors temperature by individual CPU. One CPU running significantly hotter than the others warrants investigation. There may be an opportunity to improve energy efficiency by optimizing software.
DIMM temperature	Monitors DIMM temperature by individual DIMM.
CPU prochot	Monitors use of thermal throttling due to the CPU reaching or exceeding its maximum safe operating temperature.

(*continued*)

www.dbooks.org

Table 7-10. (*continued*)

Type	Description
CPU thermtrip	Monitors use of thermal protection mechanisms. In this case, the system was powered down to prevent hardware damage due to the temperature exceeding catastrophic levels.
Memhot	Monitors memory controller use of bandwidth throttling due to one or more DIMMs reaching or exceeding their maximum safe operating temperature.
VRhot	Monitors VRs reaching or exceeding their maximum safe operating temperature.
Fan tachometer	Monitors fan speed in RPM by individual fan.
Volumetric airflow	Metric describing the volumetric airflow as a function of fan speed in RPM and the number of platform zones.
CPU utilization	This NM compute usage per second (CUPS) metric monitors average utilization across all cores. Useful for a variety of datacenter management and orchestration functions, such as VM placement.
Memory utilization	This NM CUPS metric monitors average memory utilization across all memory channels.
I/O utilization	This NM CUPS metric monitors average I/O utilization of PCIe.
Overall utilization	A composite metric, this monitors server utilization using a weighted average of the CPU, memory, and I/O CUPS utilization metrics. This provides an overall assessment of workload performance and availability indicators. It is useful for resource optimization of power and cooling in a datacenter.

The usefulness of these events is greatly enhanced when several related events are compared together. For example, monitoring the combination of PSU input and PSU output power enables users to calculate both PSU efficiency and power conversion losses.

$$Power\ \ Conversion\ Losses = Pin - Pout$$

$$PSU\ \ Efficiency = \frac{Pout}{Pin}$$

■ **Note** In calculating PSU efficiency, it is best to use Pout and Pin values averaged over a longer time window. This method takes into account the capacitance of the PSU and results in more representative values.

When BMC monitoring is measured with a workload representative of production use, it enables users to build a deeper understanding of the interactions between various components. Figure 7-1 illustrates various system component temperatures across a range of server load.

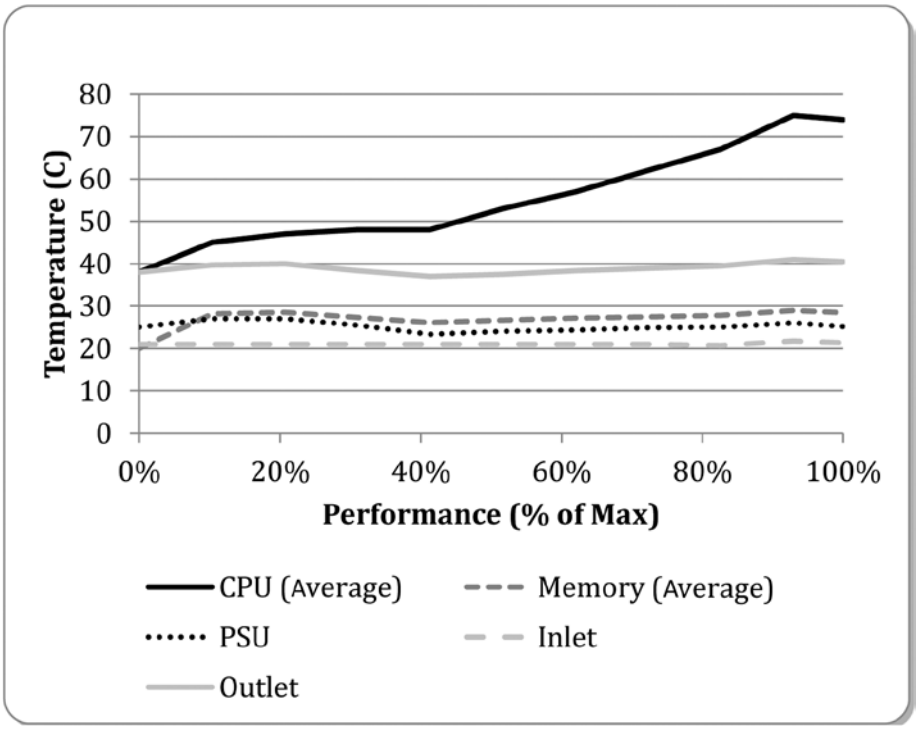


Figure 7-1. *Component temperatures across a range of server load*

The component temperature in Figure 7-1 can be compared with Figure 7-2, collected at the same time. Comparison of these figures illustrates how fan speed is increased to keep each component within a safe operating temperature. Analyzing related thermal events in conjunction with component-level power allows operators to gain insight into the efficiency of their cooling solution.

www.dbooks.org

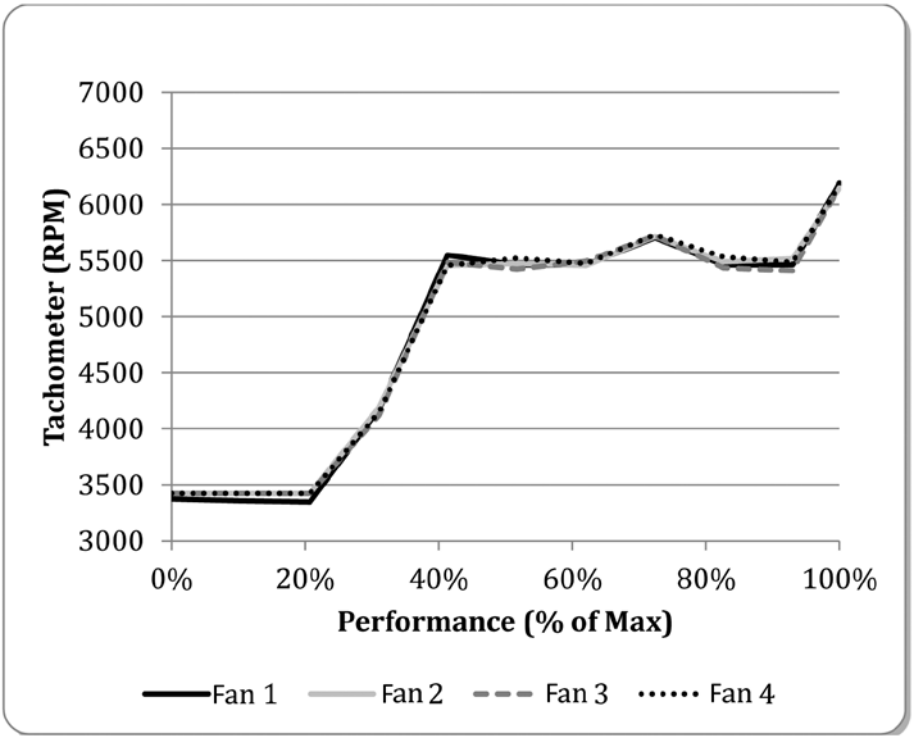


Figure 7-2. *Fan speed across a range of server load*

Software Monitoring

Chapter 6 discussed the role operating systems play in selecting power states and how operating systems balance power and performance in managing system resources. Different types of applications running on a system can pose a variety of challenges in scheduling, memory management, and I/O. Operating systems provide comprehensive monitoring capabilities that allow users to analyze this behavior. This analysis enables users to gauge how efficiently their system is running, detect poor application behavior, and discover issues with hardware configuration.

Operating systems track many of the same events that are monitored by the CPU. For example, the operating system and CPU can both monitor kernel and user time. In some cases, the operating system is able to provide unique insight into events not understood by the CPU. Operating systems can track time spent in system calls, time spent in interrupt handlers, and time spent submitting and completing I/O. They can also track time spent by individual processes. These enrichments allow the operating system to provide deeper insight into events such as kernel time. In some cases, the operating system is able to add a different perspective to events measured by both the CPU and operating system. For example, the CPU monitors P-state residency in terms of the states

that were granted. The operating system is capable of monitoring P-state residency both in terms of the states that were granted and the states that were requested.

When operating system events are used in conjunction with the CPU events, users can build a deeper understanding of the software/hardware interface. For example, the CPU is capable of measuring a specific effect, such as a core being idle 90% of the time and using only C1. The operating system is capable of measuring a specific cause, such as frequent network interrupt handling on the core with high C1 residency.

Another difference between events measured by the CPU and the operating system is accuracy. CPU events are typically clock-cycle accurate, whereas the accuracy of operating system events can vary between products and versions. For example, some events may only be sampled instead of measured, and some events may only be updated during infrequent clock interrupts.

Events used to monitor resource utilization and kernel functions are common across different operating systems. Although the events themselves are similar, there can be some subtle differences in the event names and in precisely what is being measured. For example, one operating system may include kernel time, queue time, and device time in its measure of drive latency, whereas another operating system may only include device time. This section discusses common operating system events in an operating system–independent fashion. Following an outline of these events, several examples of different operating system–specific tools and usages are provided.

Utilization and Processor Time

The operating system is capable of breaking down active and idle time into a very detailed set of information. These events can be used to determine how much time is spent executing application code to identify applications that are running at unexpected times or to identify applications that are running more frequently than expected. For example, these events can detect an intrusive management or security service that may be keeping the system out of a low power idle state.

Processor time events can be analyzed across logical processors, across cores, or across packages to identify utilization asymmetry. This may indicate misconfigured software or legacy software with poor parallel design that is leading to inefficient operation. These events can also be used to assess resource utilization as activity increases or decreases over time. Several of the charts in this chapter illustrate this type of example, demonstrating how an event changes across the full range of server load. Table 7-11 lists several common events describing processor time.

www.dbooks.org

Table 7-11. *Common CPU Utilization Events Exposed by Operating Systems*

Type	Description
User time	Monitors the time spent executing application code. Several operating system tools exist, such as Perfmon on Windows environments or SAR on Linux environments, to break down user time by thread, processor, or VM.
Kernel time	Also known as *privileged* time, this monitors the time spent by the operating system including scheduling, memory management, and interacting with different devices in the system.
Kernel interrupt time	Monitors the time spent processing hardware interrupts. This represents the higher priority portion of an interrupt that requires immediate attention, or the *top half*.
Kernel soft interrupt time	Also known as *software interrupt time*, *softirq time*, or a *deferred procedure call*, this monitors time spent processing the remaining lower priority operation of an interrupt, or the *bottom half*.
Kernel idle time	Monitors the time spent where there were no processes scheduled or ready to run. Some operating systems support *iowait*, a more specific kernel idle time metric that differentiates idle time between idle with or without I/Os outstanding.
Guest time (VMM only)	Monitors the time spent running guest VMs for virtualized environments.
Wait time (VMM only)	Monitors the time spent waiting for contended physical resources in virtualized environments. Wait time is also known as *steal*, *dispatch,* or *ready* time. High values can indicate oversubscription or VMs that frequently wait on preemption of another VM.

Figure 7-3 illustrates how the various components of processor time change with increasing server load. At the maximum throughput level, only 85% of time is spent executing application code, with the remaining processor time spent performing common kernel functions such as executing system calls or handling interrupts. Optimizations that decrease system time can yield significant improvements in energy efficiency as it provides additional processor time for applications completing work. For example, enabling interrupt coalescing in a network device can reduce the total number of interrupts, thus reducing system time.

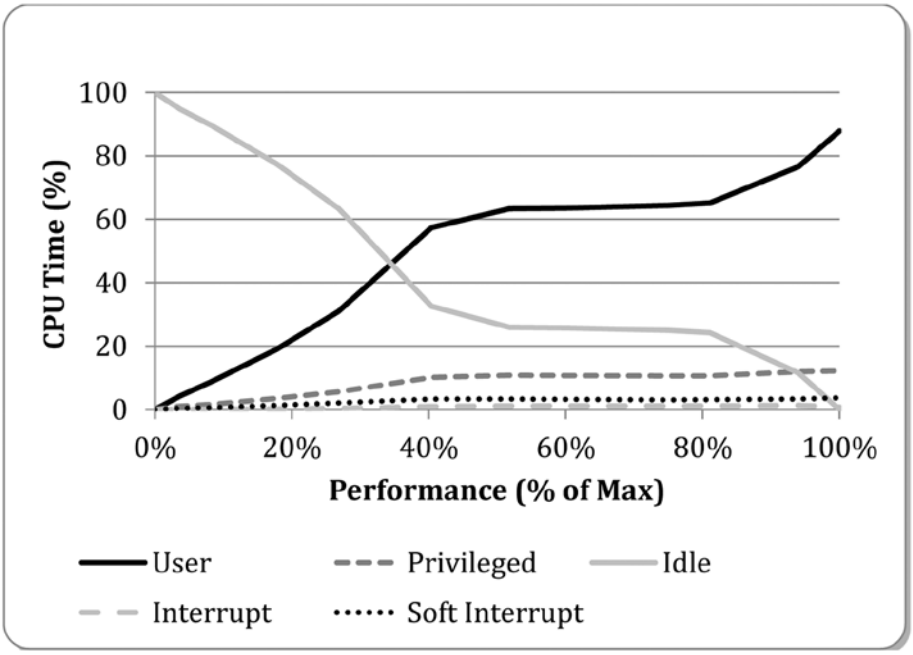


Figure 7-3. *CPU time broken down into various categories*

Simultaneous Multithreading (SMT)

When discussing processor time, it is important to revisit the relationship between physical processors and logical processors, or hardware threads in Hyper Threading (HT). Since two logical processors share the cache and execution units of a physical processor, it is possible to drive a physical processor to 100% utilization even when each of the logical processors only runs at 50% utilization. It's also possible for two logical processors at 50% utilization to only drive physical processor utilization slightly above 50%. For datacenter workloads, it's common to see physical processor utilization up to 50% higher than the reported logical processor utilization. Figure 7-4 illustrates a typical case where actual physical processor utilization is significantly higher than the logical processor utilization.

www.dbooks.org

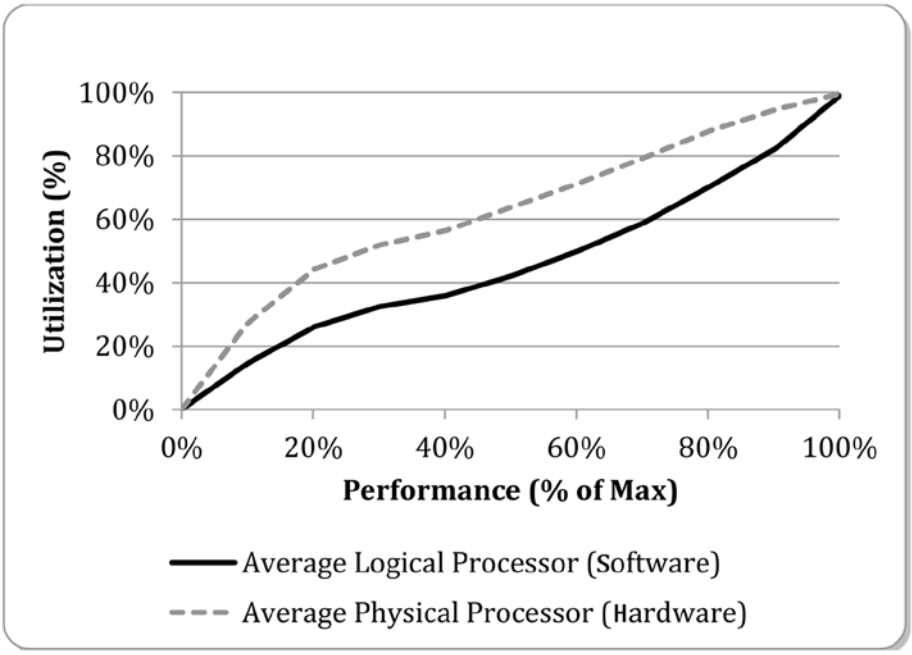


Figure 7-4. *CPU time compared between logical and physical processor utilization*

Operating systems measure utilization for logical processors, which can give a misleading picture of hardware resource utilization as power consumption, and the use of active and idle power management features are much more closely tied to physical processor utilization. Overall utilization is best measured using the hardware mechanisms described earlier in this chapter.

Virtualization

Virtualization provides several additional challenges in monitoring processor time. If utilization is measured from a virtual machine, it is a measure of virtual processor utilization, or utilization of only the resources made available to that VM. For example, in the case of VM oversubscription, it's possible for VMs to measure an average virtual processor utilization of 10%, whereas the processors themselves are running much higher than that. As a result, it would be inaccurate to conclude that a system is lightly utilized because the average virtual processor utilization is low.

Similar to the insight that can be gained by comparing physical processor utilization to logical processor utilization, additional insight can be gained be examining virtual processor utilization. For system-level analysis, monitoring is best done from the host perspective. The host has the same monitoring visibility as a native operating system as well as the ability to track individual utilization of various guest VMs.

Processor Power State Requests

Chapters 2 and 4 introduced the idea that not all operating system requests for a particular C-state or P-state are necessarily granted by hardware. For example, the operating system might request a high-frequency P-state and end up getting a lower frequency P-state due to a thermal event. Operating systems have the ability to monitor their own internal state requests in addition to what is granted by hardware. Figures 7-5 and 7-6 illustrate the differences between software-requested states and hardware-granted states in C-state residency. In Figure 7-5, ACPI C2 (hardware C6) is requested across the range of server load, but Figure 7-6 shows it is only actually used between 0% and 20% load.

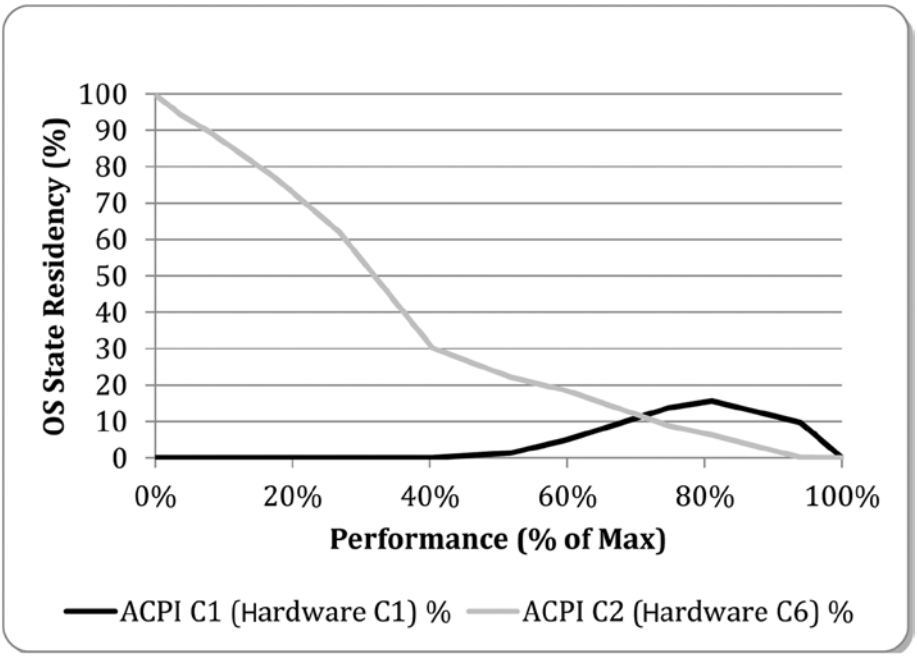


Figure 7-5. *Comparison of software C-state residency (requested) for ACPI C1 and ACPI C2*

www.dbooks.org

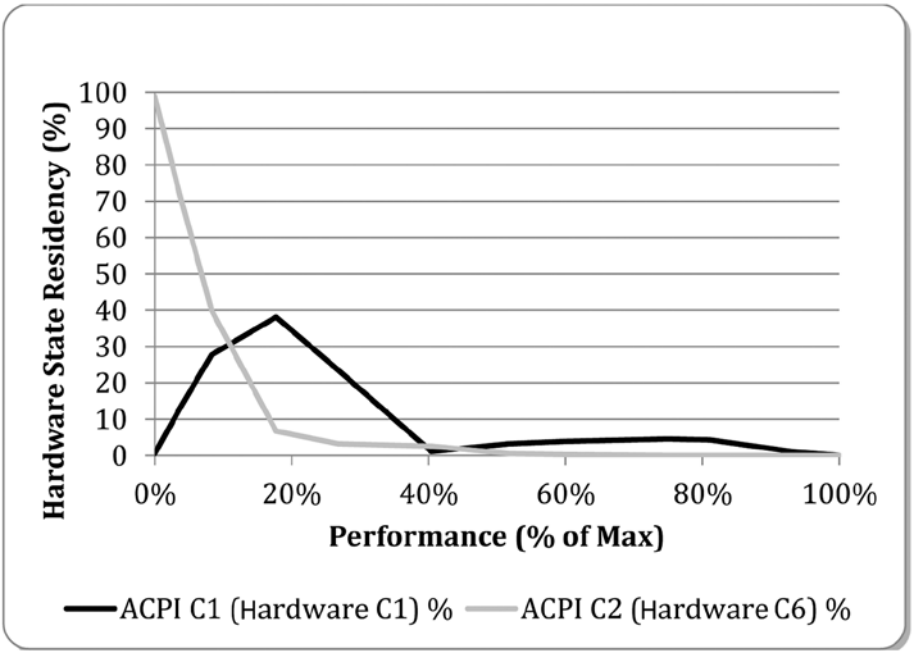


Figure 7-6. *Comparison of hardware C-state residency (granted) for ACPI C1 and ACPI C2*

■ **Note** In Figures 7-5 and 7-6 the sum of ACPI C1 (hardware C1) and ACPI C2 (hardware C6) residency adds up to total idle time. The remaining time not represented in the figure is active time.

Comparing requested residencies to granted residencies can highlight the effectiveness of various hardware and software control policies. These comparisons identify the source of unexpected state residencies, they illustrate the impact of P-state and C-state coordination between threads, and they can help guide server tuning decisions.

One reason for the substantial differences between requested and granted residencies is that software is monitoring requests at the logical processor level, whereas hardware is monitoring at the physical processor level. If half of the logical processors in a system are requesting C6 and the other half of the logical processors are requesting C1, it is possible that every core is in C1. Any time sibling logical processors are requesting different states, the shallower of the two states is granted.

Another reason for the differences between requested and granted residencies is that hardware is utilizing mechanisms to restrict use of deep C-states where it detects significant latency impact or if the energy cost to enter and exit deep C-states will not be recovered by short idle durations. These mechanisms are described in detail in Chapter 2.

Tables 7-12 and 7-13 list several common events exposed by operating systems to monitor C-state and P-state residency and transitions.

Table 7-12. *Common C-State Events Exposed by Operating Systems*

Type	Description
C1/C2/C3 residency % (software requested)	Monitors the percent of time the operating system is requesting ACPI C1, ACPI C2, or ACPI C3. This is typically measured for each logical processor.
C1/C2/C3 residency % (hardware granted)	Monitors the percent of time CPU cores actually spent in various states. Operating systems map the output of the residency MSRs described earlier in this chapter to the appropriate ACPI state.
C1/C2/C3 transitions	Monitors the number of software requests made for each ACPI C-state type. Operating system C-state transition counts are typically much higher than actual hardware C-state transitions due to the coordination of logical processors.
Average idle time	Monitors the average logical processor idle duration.
wakeups	Also known as *idle break events*, these events count the number of times a logical processor was woken up due to an interrupt or break event. Useful for assessing latency impact of C-state transitions.
Idling status	Also known as *core parking status*, this event indicates that a logical processor is not being made available for process scheduling. This is an indication of execution consolidation, discussed in Chapter 4.

Table 7-13. *Common P-State Events Exposed by Operating Systems*

Type	Description
Frequency transitions (software requested)	Monitors the number of times various operating frequencies were requested. It is measured for each logical processor. Useful for assessing the latency impact of P-state transitions.
Frequency residency % (software requested)	Monitors the percent of time the operating system spent requesting various operating frequencies. It is measured for each logical processor. For servers with a single P-state shared by all cores on same package, this can be used to identify the application or thread that drives frequency higher.

(*continued*)

www.dbooks.org

Table 7-13. (*continued*)

Type	Description
Operating frequency (hardware granted)	Monitors the operating frequency by sampling a hardware feedback mechanism that indicates the current frequency at which any logical processor is running.
TSC frequency	Also known as *CPU base frequency*, this monitors the maximum guaranteed frequency, or P1.
Maximum frequency %	Monitors the ratio of current operating frequency divided by CPU base frequency. Values greater than 1 indicate use of turbo, or non-guaranteed frequency. Useful for determining how much additional frequency turbo is granting at any given time.
Processor capacity %	Also known as *% processor utility*, this monitors the performance capacity concept introduced in Chapter 4.

Scheduler, Processes, and Threads

The scheduler's decisions in determining how compute resources are allocated, shared, and utilized play a key role in energy efficiency. Monitoring this behavior allows users to gain insights into the interaction and impact of running multiple VMs, applications, or processes concurrently. For example, if an operating system migrates processes too aggressively, the additional time it takes to restore execution context or the additional time it takes to reference remote memory can increase power. If the operating system migrates processes too conservatively, it can cause scalability issues and utilization asymmetry. Cases where one subset of logical processors is running at significantly higher utilization than the other logical processors lead to more aggressive use of higher voltage and frequency states, increasing power.

Chapters 2, 3, 4, and 6 introduced a number of decisions hardware and software power management policies need to make in order to strike the right balance between low power and low latency. The scheduler has similar challenges with similar impacts in balancing between high throughput and low latency. Scheduling decisions that minimize latency can improve transaction response times, but it can come at the cost of energy efficiency. If maximum throughput is decreased to improve latency, it results in a greater number of resources (and power) required to meet a peak performance requirement.

Operators that characterize and understand the behavior of the scheduler, processes and threads can uncover opportunities to tune thread affinity and priority to improve energy efficiency. Table 7-14 lists several common events for monitoring the scheduler.

Table 7-14. *Common Scheduler, Process, and Thread Events Exposed by Operating Systems*

Type	Description
Processor queue length	Also known as *processor queue depth*, this monitors the queue length of tasks waiting to be scheduled. Useful in conjunction with other processor time and interrupt events to identify the cause of utilization asymmetry.
Context switches	Monitors the number of times execution context was switched between processes. Useful for efficiency analysis because saving and restoring context introduces additional overhead.
Migrations	Monitors the number of times a process or thread was scheduled on a logical processor that is different from the last time.
System calls	Monitors the number of requests for the kernel to perform some action on behalf of an application, such as reading or modifying inaccessible data or interacting with hardware devices. Useful since high system call rates may indicate inefficient use of kernel interfaces.
Processes/Threads	Monitors the current number of processes and threads.
Process/Thread state	Monitors the current state of processes and threads. Provides insight on priority, readiness to run, and reasons threads are waiting.

Interrupts

The frequency of interrupts, the distribution of interrupts across logical processors, the division of interrupt processing between top and bottom halves, and the batching of interrupts provide deeper insight into the distribution of work on a server and how the interrupt processing can affect energy efficiency. For example, when clock or device interrupts occur during idle time, they cause logical processors to exit C-states. A high interrupt rate at low throughput is undesirable because low throughput typically coincides with low utilization. When interrupts occur during active time, they cause processes and threads to be suspended until processing of the interrupt is complete.

Splitting the top from the bottom half of interrupt processing enables the kernel to parallelize interrupt processing when a single logical processor is handling interrupts of a specific type. However, the bottom half doesn't necessarily execute on the same logical processor that handled the interrupt. Interrupts being processed by a very small number of logical processors can be undesirable. This can drive utilization significantly higher on logical processors handling interrupts and cause the top and bottom half of interrupts to be handled by different logical processors. This introduces additional overhead in scheduling and in accessing shared data that is not resident in one of the

www.dbooks.org

logical processor's local caches. The distribution of hard and soft interrupts can increase the overall number of interrupts due to additional IPIs.

Table 7-15 lists events that can be used to identify how interrupt handling is divided across logical processors.

Table 7-15. *Common Interrupt Events Exposed by Operating Systems*

Type	Description
Device interrupts	Monitors the number of device interrupts. Useful to monitor by specific IRQ and where interrupt processing occurs. A single device may have multiple IRQs that are handled by different logical processors.
Device soft interrupts	Also known as *softirq* or *deferred procedure call rate*, this monitors the number of software interrupts and where they occur. Useful for understanding whether the top and bottom half of interrupt handling are occurring on the same logical processor or if a logical processor is overloaded by interrupt handling.
Clock interrupts	Monitors the number of clock interrupts and where they occur. Useful to understand if clock interrupts may be impacting either C-state residency or the frequency of scheduling decisions.
IPI (inter-processor interrupts)	Monitors the number of inter-processor interrupts used to communicate between logical processors and where they occur. These are used for flushing caches and translation lookaside buffers (TLBs), for scheduling, and for requesting some action from a remote logical processor.
Interrupt coalescing	Also known as *interrupt moderation* or *interrupt batching*, this monitors hardware interrupts that are batched and processed periodically rather than when they would normally be processed. This lowers the overhead of processing interrupts but increases transaction response time. Useful for determining if default behavior for processing interrupts is biased toward energy efficiency or low latency.

Memory

It is critical to monitor both memory usage and locality to determine how an application's use of memory impacts energy efficiency. Sizing memory capacity to meet, but not exceed application requirements is critical for energy efficiency. If there is an excess of free memory capacity in the system, a significant amount of power consumption comes from memory that provides no performance benefit. Similarly, if there is not enough free memory in the system, performance and efficiency can be crippled by swapping.

Monitoring can also help determine the effectiveness of memory in use. Some applications can utilize a virtually unlimited amount of memory, but it may not be beneficial to do so. For example, applications that use memory as a cache for content stored on drives frequently hit a point of diminishing returns. At this point, use of additional memory only yields minor increases in cache hit rates, trading off a very small performance increase for a large increase in power.

Minimizing the amount of memory references that target a remote processor can provide substantial efficiency improvements. In systems with non-uniform memory access (NUMA), or systems with multiple processor sockets, each processor has faster access to local DRAM than it does to remote DRAM or DRAM attached to different sockets. Many applications aren't properly optimized for NUMA, which results in an equal amount of local and remote memory accesses. It takes more processor time to complete an operation using remote memory than it does using local memory because the increase in memory latency is reflected in CPU stall cycles.

■ **Note** It is surprising to see environments that apply extensive and aggressive efficiency optimization techniques, yet they continue to use applications not optimized for NUMA. Application NUMA optimization remains one of the more common missed opportunities for improving performance and energy efficiency, especially given that the improvements can be realized without any hardware changes.

Table 7-16 lists common events that can be used to identify how effectively memory is being utilized and to understand the locality of memory references.

Table 7-16. *Common Memory Events Exposed by Operating Systems*

Type	Description
NUMA locality	Monitors the percent or amount of memory references that are satisfied by local memory. Useful in understanding how well optimized software is for a multi-socket system. This can be collected with tools such as *NumaTOP* for Linux. NUMA locality can have a significant impact on CPU utilization.
Total memory	Monitors the total memory capacity of the system.
Used/Free memory	Monitors the amount of memory in the system currently being used. It is useful to monitor this by specific applications, processes, and threads to understand if there are areas for improvement. It is also useful to understand how much memory is being used for drive caching, since that memory appears as used but is still available for application use.

(*continued*)

www.dbooks.org

Table 7-16. (*continued*)

Type	Description
Paging	Monitors blocks or pages of memory moved in and out of physical memory from a secondary storage device.
Swapping	Monitors entire process memory footprints moved in and out of physical memory from a secondary storage device. Swapping has a severe impact on performance and energy efficiency. If a system is swapping, the workload needs to be optimized to decrease the working set size, or more memory capacity needs to be added.

I/O

Issues due to insufficient I/O performance are critical to identify because they result in the use of more servers (and more power) than necessary to meet performance requirements. I/O bottlenecks can prevent applications from being able to fully utilize CPUs and memory, causing components in the system to consume a significant amount of energy while doing little useful work.

Understanding what is sufficient in terms of I/O performance can be a significant challenge. There is a tremendous range in peak performance between different technologies available today. For example, storage subsystems can use different interfaces (3 Gb/s, 6 Gb/s, or 12 Gb/s), different protocols (SATA, SAS, or FC) and different drive types (HDD or SSD). SSDs can have a tremendous impact on system behavior by removing a latency bottleneck that plagues many workloads. In addition to the technologies being used, peak performance is dependent on I/O type, block or packet size, the mix between reads and writes, or the mix between random and sequential I/O. Operating system monitoring features are key to understanding specific workload characteristics and the limitations of an I/O subsystem.

When monitoring I/O it is important to understand the maximum performance of the I/O subsystem when compared to the necessary performance requirements. This applies to both networking and storage. Figure 7-7 compares peak drive I/O operations per second (IOPS) and drive bandwidth to runtime measurements across a range of server load.

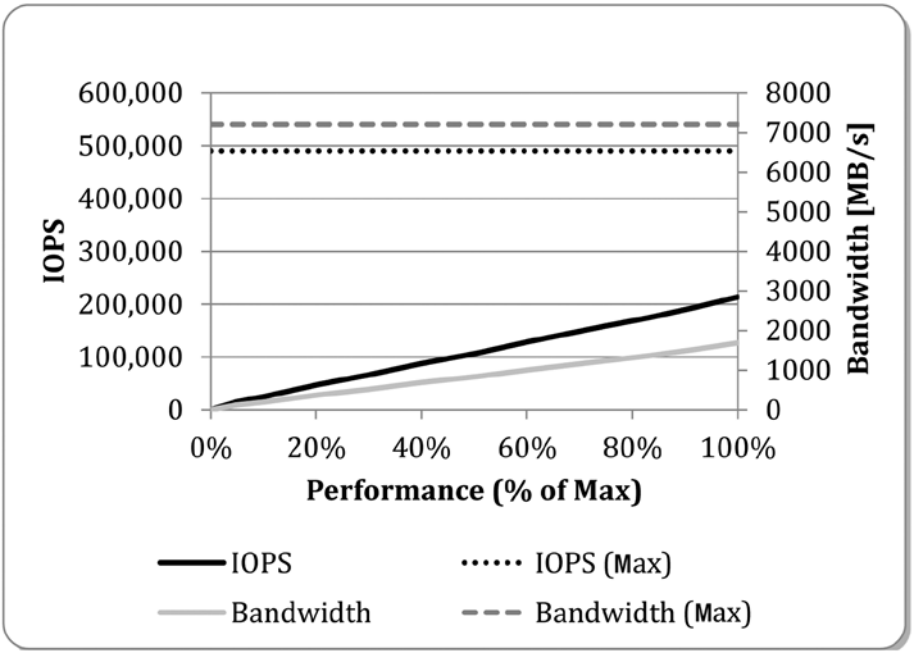


Figure 7-7. *Comparing runtime IOPS and bandwidth to theoretical maximums*

I/O bottlenecks are frequently introduced during new technology transitions. CPU and memory performance increase at a very different rate than I/O subsystem performance does. Upgrading to the latest platform may result in very different increases in compute performance compared to I/O performance. Another transition that frequently introduces issues with insufficient I/O performance is virtualization. With several VMs sharing an I/O subsystem, increases in traffic, in resource competition, and in diversity of I/O traffic can cause significant decreases in peak I/O performance.

Some I/O bottlenecks can be addressed through tuning, for example, enabling offloading capabilities in an I/O adapter, segmenting or segregating traffic to specific interfaces, enabling interrupt batching, or using virtualization technologies for directed I/O (VT-d). Table 7-15 lists common events that can be used to monitor I/O performance and to compare runtime performance to peak performance capabilities.

www.dbooks.org

Table 7-17. *Common I/O Events Exposed by Operating Systems*

Type	Description
Reads/Writes (or Rx/Tx)	Monitors the number of reads and writes. Useful for calculating IOPS.
Read/Write Bytes (or Rx/Tx Bytes)	Monitors the bandwidth of reads and writes. Useful for monitoring on a per-drive or per-interface level to pinpoint potential issues.
Queue length	Monitors the average queue length for reads and writes or the average number of I/Os waiting to be processed. Useful for identifying I/O bottlenecks.
Queue wait time	Also known as *queue latency*, this monitors the average time I/O requests wait in a queue before they are submitted to a device. Useful for identifying I/O bottlenecks.
Service time	Also known as *device latency*, this monitors the average time it takes for an I/O submitted to a device to be completed. Useful in combination with queue wait time to understand how different phases of I/O contribute to end-to-end latency.
Latency	Monitors end-to-end latency of an I/O including kernel time.
Utilization %	Monitors the % of time a device is active processing I/Os.
Controller idle states	Monitors the device power states of various controllers.

Tools

This chapter introduced several low-level mechanisms for configuring and accessing monitoring features. For most uses, this complexity can be managed by software tools rather than by an end user. The following section provides a short description of some common software tools and sample usages. This is not intended to be a comprehensive list of all tools and usages. Rather, it introduces the reader to the type of tools available for monitoring and how they can be used. Extensive documentation for these software tools is available online.

Health Checks

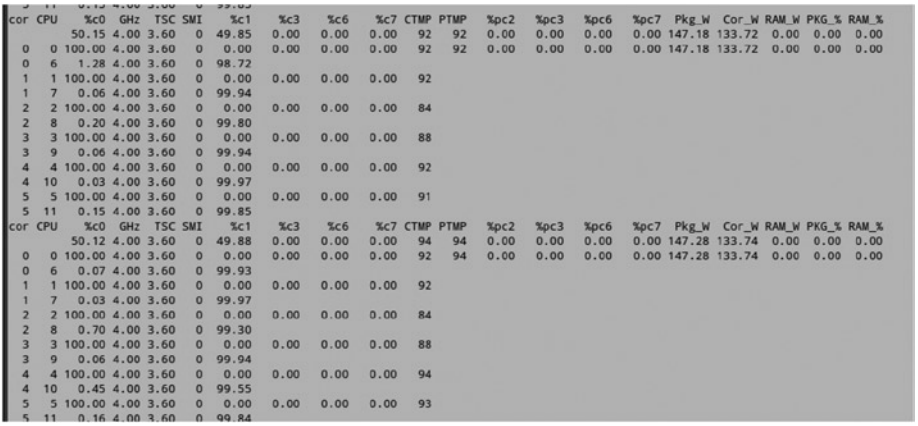
Many times, users are interested in getting a high-level picture of what is going on in the system and are less interested in diving into the architectural and micro-architectural details. There are two common tools for Linux (*PowerTOP* and *turbostat*) and two for Windows (*Perfmon* and *Powercfg*) that provide an excellent first stop for information about the power characteristics of a system.

Turbostat (Linux)

Turbostat is a simple but powerful tool that is built into the Linux kernel tree. It monitors

- Per-thread: Average frequency, activity
- Per-core: Core C-states, temperature
- Per-package: Temperature, package C-states, package power, core power (where supported), DRAM power

Turbostat has several different command-line options that can come in handy for a range of usage models. Simply running it without any parameters will provide one-second snapshots of a range of statistics. Figure 7-8 shows an example of turbostat output.



```
cor CPU    %c0  GHz  TSC SMI    %c1    %c3    %c6    %c7 CTMP PTMP   %pc2   %pc3   %pc6   %pc7  Pkg_W  Cor_W RAM_W PKG_% RAM_%
         50.15 4.00 3.60   0  49.85   0.00   0.00   0.00   92   92   0.00   0.00   0.00   0.00 147.18 133.72  0.00  0.00  0.00
  0   0 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   92   92   0.00   0.00   0.00   0.00 147.18 133.72  0.00  0.00  0.00
  0   6   1.28 4.00 3.60   0  98.72
  1   1 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   92
  1   7   0.06 4.00 3.60   0  99.94
  2   2 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   84
  2   8   0.20 4.00 3.60   0  99.80
  3   3 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   88
  3   9   0.06 4.00 3.60   0  99.94
  4   4 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   92
  4  10   0.03 4.00 3.60   0  99.97
  5   5 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   91
  5  11   0.15 4.00 3.60   0  99.85
cor CPU    %c0  GHz  TSC SMI    %c1    %c3    %c6    %c7 CTMP PTMP   %pc2   %pc3   %pc6   %pc7  Pkg_W  Cor_W RAM_W PKG_% RAM_%
         50.12 4.00 3.60   0  49.88   0.00   0.00   0.00   94   94   0.00   0.00   0.00   0.00 147.28 133.74  0.00  0.00  0.00
  0   0 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   92   94   0.00   0.00   0.00   0.00 147.28 133.74  0.00  0.00  0.00
  0   6   0.07 4.00 3.60   0  99.93
  1   1 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   92
  1   7   0.03 4.00 3.60   0  99.97
  2   2 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   84
  2   8   0.70 4.00 3.60   0  99.30
  3   3 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   88
  3   9   0.06 4.00 3.60   0  99.94
  4   4 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   94
  4  10   0.45 4.00 3.60   0  99.55
  5   5 100.00 4.00 3.60   0   0.00   0.00   0.00   0.00   93
  5  11   0.16 4.00 3.60   0  99.84
```

Figure 7-8. *Turbostat output from a 3.12 kernel*

Turbostat is commonly run alongside a workload to get statistics about the system during the measurement. Note that although the statistics provided are heavily influenced by the workload, it is also affected by anything else running on the system.

```
>> turbostat <program>
```

The `-v` option is a great way to collect a wide range of debug information about the system configuration.

```
>> turbostat -v
```

Running turbostat at one-second intervals, particularly on large multi-socket systems, can perturb workload behavior (and the results from the tool). Consider increasing the monitoring interval if statistics at one-second intervals are not necessary. This is particularly useful when trying to collect statistics about an idle system.

```
>> turbostat -i 2
```


www.dbooks.org

Turbostat can also be used as a simple tool for monitoring various MSRs in the system. The -m and -M options will read the corresponding MSR one time from each thread for each sample. The -M option will dump a 64-bit output, whereas the -m option does a short output. The following command will dump MSR 0x199 at two-second intervals.

```
>> turbostat -i 2 -M 0x199
```

The -C and -c options provide a similar capability, but instead of displaying the raw value of the register, these options provide a dump of the delta between the current sample and the previous sample. This can be useful for monitoring the deltas for counters over time. As an example, the following command will dump the delta in APREF (MSR 0xE7) every one-second sample.

```
>> turbostat -i 1 -M 0xe7
```

Not all distributions automatically include the turbostat binary, but it is trivial to compile and use once you have the kernel source. The tool is stand-alone, so if you download a recent kernel (from kernel.org), it will include the source. Turbostat requires that your kernel have MSR support.[3]

```
// Extract the kernel source after downloading it
>> tar xf linux-3.12.20.tar.xz

// find the sourcecode and cd to the relevant directory
>> cd linux-3.12.20
>> find . -name turbostat.c
>> cd tools/power/x86/turbostat/

// build the tool
>> make

// make sure that the MSR kernel module is active
>> sudo modprobe msr

// run the tool
>> sudo ./turbostat
```

PowerTOP (Linux)

PowerTOP is an open-source tool for characterizing power management and diagnosing power management issues. Like turbostat, it is targeted at various usage models (not just servers). It is a useful addition because it provides some additional information above and beyond what turbostat provides. Some notable additions include average time in

[3]If your kernel was configured without MSR support (either built in or through a kernel module), then you will need to recompile your kernel in order to use turbostat.

C-states and frequency histograms. Powertop also provides a large amount of information targeted at consumer usage models (device idle power). The device statistics tend to be less relevant on servers. Figure 7-9 shows sample output of the tool.

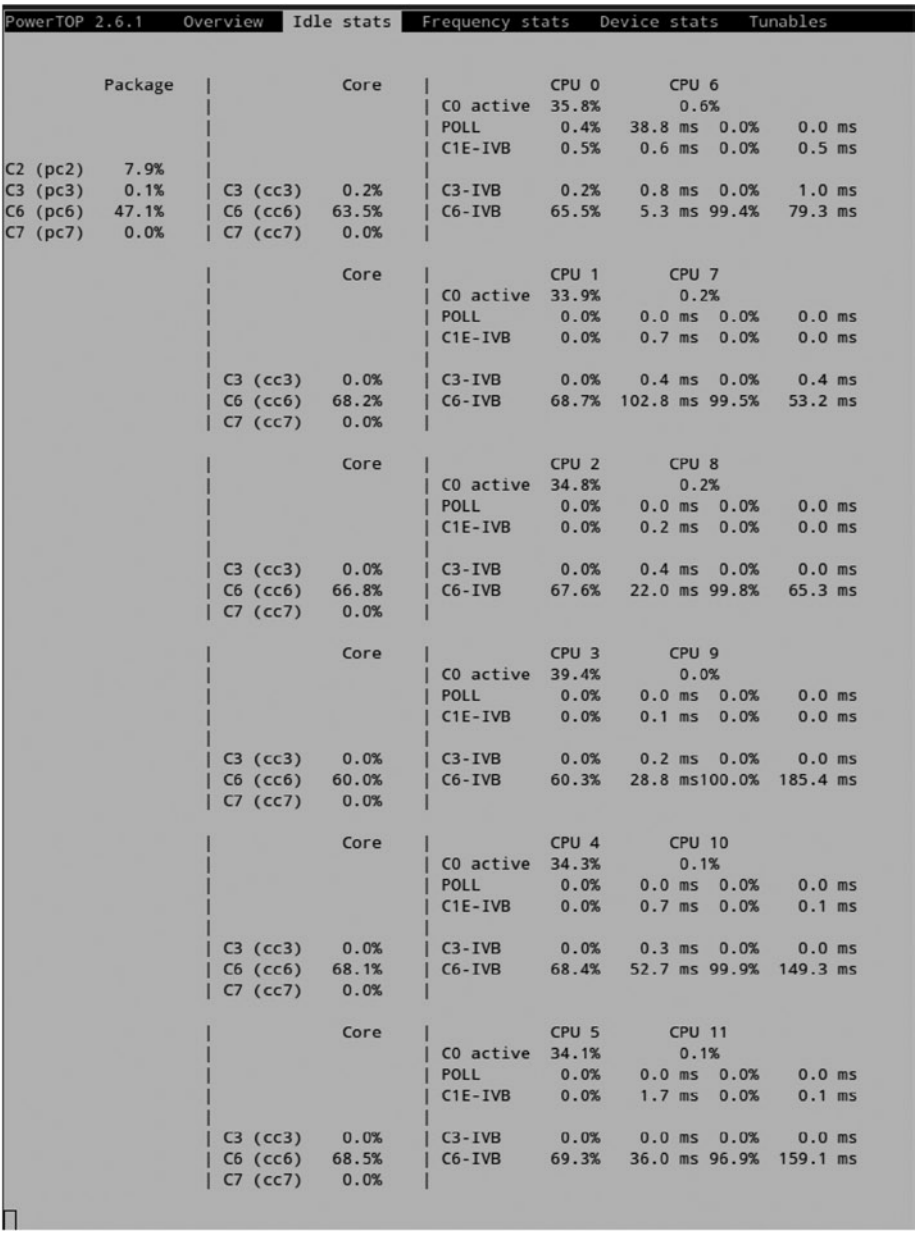


Figure 7-9. *Powertop 2.6.1 idle stats*

www.dbooks.org

The basic powertop command line provides slow sample intervals by default (many seconds). This can be useful to minimize the application's overhead, particularly on idle systems, but it also provides much slower results. The time parameter can speed this up (at the expense of increased CPU overhead).

The `--html` option will dump an HTML file. By default, this will collect a single measurement of the tool. However, more measurements can be collected, generating multiple HTML files. The `--html` option can collect statistics over the execution of a workload with the `--workload` parameter. Similar to turbostat, powertop will collect statistics for the entire system and not just the specified workload.

```
>> powertop --workload=./test.sh --html=test.powertop.html
```

Powercfg (Windows)

Powercfg is a Windows command-line tool that enables users to tune lower-level operating system power management settings. It allows users to enable and disable features, change power policies, and identify issues that may impact power management. For example, Powercfg can be used to change settings for hard drive power options during inactivity and to query devices to understand the power states they support.

One of the unique applications of Powercfg is its ability to generate an energy report. This option analyzes the system and reports events and configuration details that may impact power management. The Powercfg energy report gives detailed statistics on idle interruption, device activity, failure of devices to support power states, changes to the operating system timer frequency, and supported power states.

The following example command uses Powercfg to list the different power policies supported by the operating system and indicates that the current active policy is balanced. This setting also corresponds to how the operating system is setting IA32_ENERGY_PERF_BIAS during initialization.

```
C:\Windows\system32>powercfg -list

Existing Power Schemes (* Active)
-----------------------------------
Power Scheme GUID: 381b4222-f694-41f0-9685-ff5bb260df2e  (Balanced) *
Power Scheme GUID: 8c5e7fda-e8bf-4a96-9a85-a6e23a8c635c  (High performance)
Power Scheme GUID: a1841308-3541-4fab-bc81-f71556f20b4a  (Power saver)
```

The following example command shows how to create the energy report.

```
C:\Windows\system32>powercfg.exe /energy
```

Figure 7-10 shows a sample of the energy report results. In this example, the system is seeing poor package C-state residency, and the energy report has identified that several USB devices are connected to the server that do not have USB selective suspend enabled. This activity is preventing the system from maximizing residency in its lowest idle power state.



Figure 7-10. *A portion of the energy report generated by the Windows Powercfg tool*

Hardware Monitoring Tools

Programming and reading the various performance counters in the core and uncore of a processor can become fairly complicated. In order to simplify this task, there are a number of stand-alone tools that perform the event programming, data collection, and visualization for the user.

Intel Performance Counter Monitor (PCM)

PCM provides stand-alone tools that handle counter programming and collection for the user. PCM also includes sample routines that demonstrate how to configure and read performance counters with open-source C++ code. These routines translate the raw events into meaningful metrics like memory traffic into GB/s or energy consumed into Joules. PCM is targeted at both power and performance monitoring and characterization and is available as source code with a BSD-like license at `www.intel.com/software/pcm`.

Intel PCM runs on multiple platforms including Linux, Microsoft Windows, FreeBSD, and Mac OS X. This is possible, because it only requires a driver to program the MSRs. For platforms like Windows, which do not already provide such a driver, the package includes sample code for the driver as well. Binaries are not distributed at the time of publishing, but instructions for compilation are included.

PCM can be used in one of two ways: (1) as a set of stand-alone utilities, or (2) integrated into an application. Figure 7-11 shows an example of a graph generated from data collected by the stand-alone pcm-power utility on a four-socket system.

www.dbooks.org

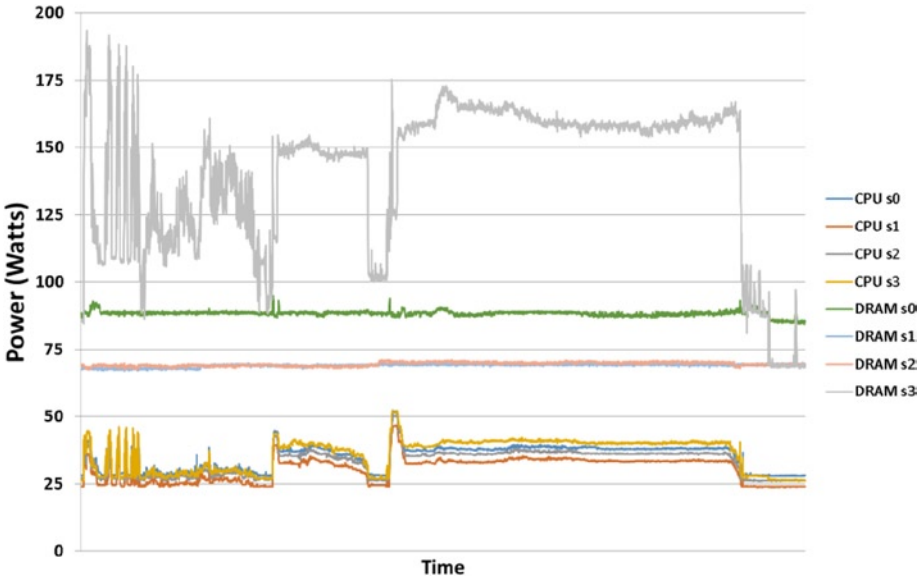


Figure 7-11. *Socket and DRAM power trace generated from PCM-collected data on four-socket system*

The standalone command-line program PCM is similar to turbostat in that it periodically prints the output to the screen for a given time interval. This time interval can be specified as the first parameter. For large servers with tens or hundreds of cores, it is often useful to suppress the metrics for individual cores by using the `-nc` parameter (for example, `> pcm.x 1 –nc`). An example from a four-socket system is show in Figure 7-12.

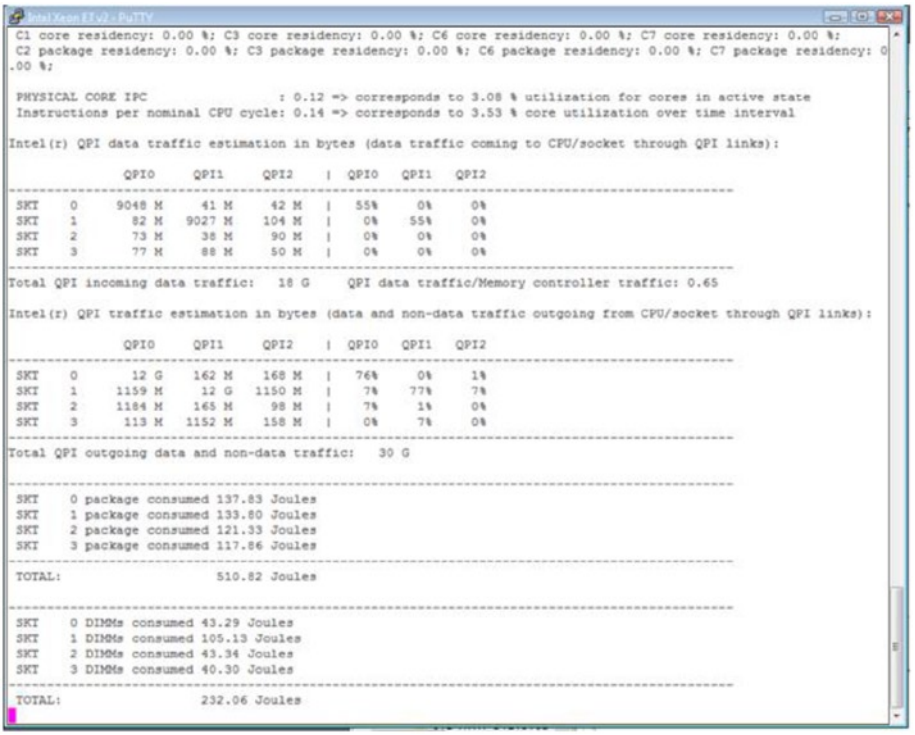


Figure 7-12. *Standalone PCM showing power and performance metrics from a four-socket system*

It is also possible to monitor with PCM throughout the duration of a workload by providing the workload executable as a parameter. Since PCM monitors the whole system, the script can actually be a workload driver for a server application that was started beforehand. Please note that in this case, PCM reports the metrics for the entire workload measurement, for example, the total energy consumption.

```
> pcm.x ./run_workload.sh
```

For simpler post-processing by spreadsheet program, another convenient feature is the ability to generate comma-separated lists using the `-csv` option:

```
> pcm.x 1 -csv 2>&1 | tee pcm.txt
```

PCM includes a utility targeted exclusively at power management: pcm-power. This utility can measure a number of the statistics made available through the uncore performance monitors on Xeon E5/E7 processors, including these:

- Core C-state residencies
- Causes of frequency throttling (thermal, power, OS requested, electrical/fuse)

www.dbooks.org

- Frequency transition statistics
- Prochot statistics
- Frequency histograms
- DRAM power savings (CKE and self-refresh)

For example, to monitor the number of frequency transitions occurring in the system at one-second intervals (and hide memory statistics), one would execute

```
> pcm-power.x 1 -p 5 -m -1
```

Figure 7-13 shows the output of this command. In addition to displaying frequency transition statistics, some power and thermal statistics are also displayed. Additional information is collected using the free-running counters and therefore is displayed regardless of the command line. Note that the DRAM Energy counter was not enabled on the system under test here and therefore reported 0. DRAM RAPL is not a required capability and is not supported on all platforms.

```
Time elapsed: 1000 ms
Called sleep function for 1000 ms
S0; PCUClocks: 800244537; Frequency transition count: 77 ; Cycles spent changing frequency: 0.09 %
S0; Consumed energy units: 549494; Consumed Joules: 8.38; Watts: 8.38; Thermal headroom below TjMax: 53
S0; Consumed DRAM energy units: 0; Consumed DRAM Joules: 0.00; DRAM Watts: 0.00
---------------------------------------------------------------------------------------------
Time elapsed: 1000 ms
Called sleep function for 1000 ms
S0; PCUClocks: 800243447; Frequency transition count: 8 ; Cycles spent changing frequency: 0.01 %
S0; Consumed energy units: 542448; Consumed Joules: 8.28; Watts: 8.28; Thermal headroom below TjMax: 51
S0; Consumed DRAM energy units: 0; Consumed DRAM Joules: 0.00; DRAM Watts: 0.00
```

Figure 7-13. *pcm-power screenshot—frequency transitions*

Figure 7-14 shows an example where pcm-power is monitoring why the system is unable to achieve the maximum possible frequency (frequency clipping cause) by using the following command line. In this command line, grep is used to filter out some of the extraneous information.

```
> pcm-power.x 1 -p 3 -m -1 | grep -E "(PCUClocks|limit cycles)"
```

```
S0; Consumed energy units: 8955233; Consumed Joules: 136.65; Watts: 136.65; Thermal headroom below TjMax: 1
S0; PCUClocks: 800131708; Thermal freq limit cycles: 90.32 %; Power freq limit cycles:0.00 %; Clipped freq limit cycles:0.00 %
S0; Consumed energy units: 8981871; Consumed Joules: 137.05; Watts: 137.05; Thermal headroom below TjMax: 0
S0; PCUClocks: 800133133; Thermal freq limit cycles: 87.79 %; Power freq limit cycles:0.00 %; Clipped freq limit cycles:0.00 %
S0; Consumed energy units: 8896222; Consumed Joules: 135.75; Watts: 135.75; Thermal headroom below TjMax: 1
S0; PCUClocks: 800132966; Thermal freq limit cycles: 90.67 %; Power freq limit cycles:0.00 %; Clipped freq limit cycles:0.00 %
S0; Consumed energy units: 8858975; Consumed Joules: 135.18; Watts: 135.18; Thermal headroom below TjMax: 0
S0; PCUClocks: 800142316; Thermal freq limit cycles: 92.02 %; Power freq limit cycles:0.00 %; Clipped freq limit cycles:0.00 %
S0; Consumed energy units: 8829438; Consumed Joules: 134.73; Watts: 134.59; Thermal headroom below TjMax: 1
S0; PCUClocks: 800142124; Thermal freq limit cycles: 91.28 %; Power freq limit cycles:0.00 %; Clipped freq limit cycles:0.00 %
```

Figure 7-14. *pcm-power screenshot—frequency clipping cause*

The "headroom below TjMax" is shown as 1 or 0, indicating that the system is at the thermal limit. At the same time, the "Thermal freq limit cycles" is hovering at 90%, indicating that the frequency of the system is being limited because of thermal limits a large percentage of the time.

A number of other targeted standalone applications are available as well:

- pcm-numa reports, for each core, the traffic to local and remote memory.
- pcm-memory reports memory traffic per memory channel.
- pcm-pcie reports memory traffic to and from PCIe devices.
- pcm-power can report multiple values depending on the parameter selection.

Both KSysGuard (KDE) and Windows Perfmon provide visualization mechanisms for monitoring individual counters in real time. See the PCM webpage for the latest recipes.

Since PCM is distributed as source code, it can also be integrated directly into an application to facilitate collecting system-wide statistics while an application executes. The initialization is as easy as

```
PCM * m = PCM::getInstance();
if (m->program() != PCM::Success) return;
```

The actual measurement is similar to measuring time, where you store the clock before and after the critical code and then take the difference. For the performance counters, there are states available per core, package (socket), and system. There are also predefined functions for all supported metrics:

```
SystemCounterState before = getSystemCounterState();

// run your code here

SystemCounterState after = getSystemCounterState();
```

Then, specific statistics can be displayed with, for example, the following:

```
cout << "Instructions: " << getInstructonsRetired(before, after)
     << "CPU Energy  : " << getConsumedEnergy(before, after)
     << "DRAM Energy : " << getDRAMConsumedEnergy(before, after);
```

Linux Perf

Newer Linux systems have an integrated profiling and tracing subsystem called perf_events. The perf_events subsystem provides an interface to the CPU's Performance Monitoring Units (PMUs); it provides an interface to the software tracepoints provided by the Linux kernel, and it takes care of sharing resources between different users. A standard command-line tool called "perf" allows access to the perf_events interface. Other tools and libraries, such as NumaTOP or PAPI, also utilize the perf_events subsystem. There are also GUI frontends available, such as Eclipse perf or sysprof. The perf tool is typically included as a package in the Linux distribution. A wiki with documentation about using perf can be found at https://perf.wiki.kernel.org/index.php/Tutorial.

www.dbooks.org

The perf_events subsystem is integrated into the Linux kernel with the functionality varying depending on kernel version. All perf versions have support for basic PMU profiling with sampling. The perf top utility (shown in Figure 7-15) is an easy way to see details about where the CPU is currently spending its time.



Figure 7-15. *Perf top example*

Perf also provides access to software trace points provided by the kernel. For example, `perf timechart record` records all schedule events and idle periods and can generate a GANTT-style chart with a perf timechart report. This is useful for understanding short stretches (a few seconds) of workload behavior. Timechart first records the system behavior to a `perf.data` and then generates a SVG file to visualize the trace in a GANTT chart–like representation.

```
% sudo perf timechart record sleep 1
[ perf record: Woken up 2 times to write data ]
[ perf record: Captured and wrote 1.289 MB perf.data (~56298 samples) ]
% sudo perf timechart
Written 1.0 seconds of trace to output.svg.
```

`output.svg` can then be viewed in a SVG viewer, for example, with Chrome. There are two sets of data provided in the SVG timecharts:

- A view of what is running on each of the logical processors (Figure 7-16)
- A view of the activity of each of the software threads (Figure 7-17)

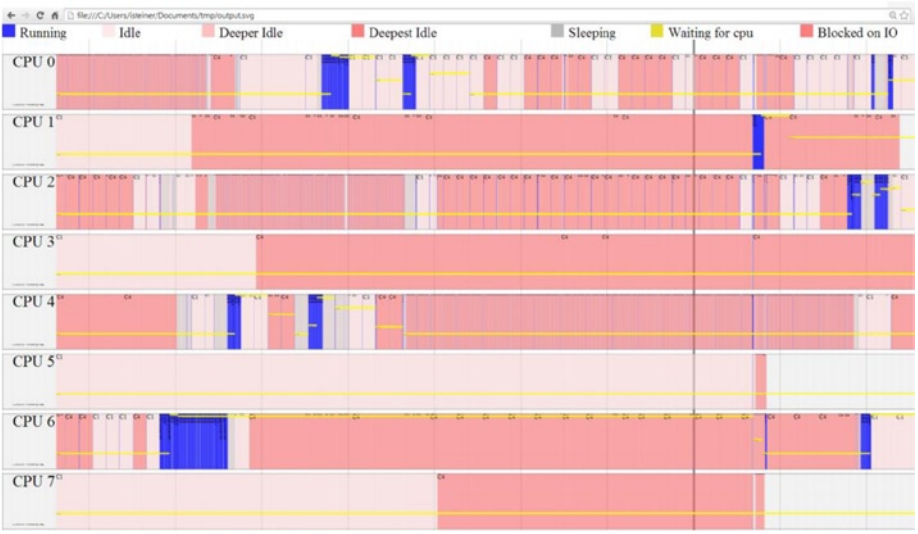


Figure 7-16. *Timechart part 1: logical processor activity*



Figure 7-17. *Timechart part 2: software thread activity*

www.dbooks.org

The logical processor information in Figure 7-16 shows both the requested C-states for each of the threads as well as the threads that are active on each of the virtual CPUs (in blue). When cores are asleep, it provides guidance about whether they are waiting for I/O (called the *io_schedule* routine) or if they are idle and waiting for the CPU (called the *schedule* routine).

The software thread view in Figure 7-17 shows when each of the different software threads are active (in blue) and when the threads are inactive (in gray). The trace will show the threads starting the first time it sees them execute (and not necessarily when they actually began execution).

In some cases it's also useful to look at the raw trace events, which can be post-processed with scripts to extract information of interest. The visual timecharts can be challenging to work with on large systems with many threads. The data file generated by the `perf timechart record` can be viewed with `perf script` in text format as an alternative to the visual timechart.

```
% sudo perf script | less
       swapper     0 [000] 176421.261802: power:cpu_idle: state=4 cpu_id=0
       perf     7667 [001] 176421.262026: sched:sched_switch: prev_comm=perf top
       swapper     0 [000] 176421.262692: sched:sched_wakeup: comm=qemu-system
       swapper     0 [000] 176421.262694: power:cpu_idle: state=4294967295 cpu
```

The first line is the process (swapper means idle), then the pid, CPU number, timestamp, event name, and event parameters. In the first line, CPU 0 goes to sleep with C-state 4. Shortly after, there is a context switch of `perf` to a thread of `top` on CPU 1. Then eventually a `qemu-system` process wakes up CPU 0, which causes an idle exit.

More trace points can be displayed with `perf list` (as root), recorded with `perf record`, and displayed with `perf script`.

A couple of useful tracing features in perf are `kprobes` and uprobes. These allow you to create new trace points dynamically in the kernel or in user programs. These can be accessed with the `perf probe` command. The following example sets a probe on the `malloc` function and measures `malloc` accesses:

```
// list functions available
% perf probe -F -x /lib64/libc.so.6  | grep malloc
Malloc

// add a trace point
% sudo perf probe -x /lib64/libc.so.6 malloc
Added new event:
  probe_libc:malloc    (on 0x82520)

// collect a trace
% perf record -e probe_libc:malloc sleep 1

// generate a report from the trace
% perf report
// remove the trace point when done
% perf probe –d probe_libc:malloc
```

The `perf stat` command can also be used to access the CPU energy meters (requires kernel 3.14+). The following example command collects package energy use on a single socket system every 100 ms. To get a break down for multiple sockets, `--per-socket` can be used.

```
$ sudo perf stat -I100 -e power/energy-pkg/              -a sleep 1
#           time             counts   unit events
     0.100177504              0.25 Joules power/energy-cores/      [100.00%]
     0.100177504              0.86 Joules power/energy-pkg/
...
     0.701274659              0.23 Joules power/energy-cores/
     0.701274659              0.82 Joules power/energy-pkg/
```

Perf has a simple built-in performance monitoring event list (see perf list). In addition, it is also possible to specify events raw (cpu/event=0x54,umask=0xFF/).

On Xeon E5/E7 processors, it may also be possible to access the uncore events through perf. This can be done with the `ucperf.py` tool in pmu-tools. This example prints the percentage of time the socket's frequency is thermally limited every second.

```
sudo ./ucevent.py  PCU.PCT_CYC_FREQ_THERMAL_LTD
S0-PCU.PCT_CYC_FREQ_THERMAL_LTD
0.00
0.00
0.00
```

IPMItool

The most frequently used tool to access BMC monitoring capabilities as well as BMC health, inventory, and management functions is IPMItool. This open source command-line tool supports out-of-band access via an authenticated network connection as well as in-band use via a device driver on the server.

The following example demonstrates use of IPMItool by listing the SDR. This command enables users to determine what sensors are available and do a quick check on the status of those sensors. In this example, all but the temperature and thermal sensors have been removed to simplify the example. Specific commands and a full list of available command-line options are available by invoking the tool's help option, and additional information can be readily found online.

```
# ipmitool -I lanplus -H xeon-bmc -U root -P pass sdr

BB P1 VR Temp    | no reading        | ns
Front Panel Temp | 22 degrees C      | ok
SSB Temp         | no reading        | ns
BB P2 VR Temp    | no reading        | ns
BB Vtt 2 Temp    | no reading        | ns
BB Vtt 1 Temp    | no reading        | ns
I/O Mod Temp     | no reading        | ns
```


www.dbooks.org

```
HSBP 1 Temp      | no reading        | ns
SAS Mod Temp     | 25 degrees C      | ok
Exit Air Temp    | no reading        | ns
LAN NIC Temp     | 35 degrees C      | ok
PS1 Temperature  | 26 degrees C      | ok
PS2 Temperature  | no reading        | ns
P1 Therm Margin  | -59 degrees C     | ok
P2 Therm Margin  | -60 degrees C     | ok
P1 Therm Ctrl %  | 0 unspecified     | ok
P2 Therm Ctrl %  | 0 unspecified     | ok
P1 DTS Therm Mgn | -59 degrees C     | ok
P2 DTS Therm Mgn | -60 degrees C     | ok
P1 VRD Hot       | 0x00              | ok
P2 VRD Hot       | 0x00              | ok
P1 Mem01 VRD Hot | 0x00              | ok
P1 Mem23 VRD Hot | 0x00              | ok
P2 Mem01 VRD Hot | 0x00              | ok
P2 Mem23 VRD Hot | 0x00              | ok
DIMM Thrm Mrgn 1 | no reading        | ns
DIMM Thrm Mrgn 2 | no reading        | ns
DIMM Thrm Mrgn 3 | no reading        | ns
DIMM Thrm Mrgn 4 | no reading        | ns
Mem P1 Thrm Trip | 0x00              | ok
Mem P2 Thrm Trip | 0x00              | ok
Agg Therm Mgn 1  | no reading        | ns
BB +12.0V        | 10.78 Volts       | nc
BB +5.0V         | -1.65 Volts       | cr
BB +3.3V         | 1.91 Volts        | cr
BB +5.0V STBY    | 2.93 Volts        | cr
BB +3.3V AUX     | 1.91 Volts        | cr
BB +1.05Vccp P1  | 1.53 Volts        | cr
BB +1.05Vccp P2  | 1.04 Volts        | ok
BB +1.5 P1MEM AB | 1.48 Volts        | ok
BB +1.5 P1MEM CD | 1.03 Volts        | cr
BB +1.5 P2MEM AB | 1.03 Volts        | cr
BB +1.5 P2MEM CD | 0.78 Volts        | cr
BB +1.8V AUX     | 1.94 Volts        | nc
BB +1.1V STBY    | 1.30 Volts        | cr
BB +3.3V Vbat    | 3.08 Volts        | ok
BB +1.35 P1LV AB | disabled          | ns
BB +1.35 P1LV CD | disabled          | ns
BB +1.35 P2LV AB | disabled          | ns
BB +1.35 P2LV CD | disabled          | ns
BB +3.3 RSR1 PGD | 3.43 Volts        | ok
BB +3.3 RSR2 PGD | 0.65 Volts        | cr
```

The following example command demonstrates use of IPMItool drill-down into fan-specific sensors. The first example shows how to read the tachometer for connected fans. There are cases where a sensor exists in the SDR, but it is currently not reporting a measurement due to being powered off, disconnected, or unsupported. Disconnected fans have been removed in this example for simplicity.

```
# ipmitool -I lanplus -H xeon-bmc -U root -P pass sdr type fan

System Fan 1A     | 30h | ok  | 29.1  | 5890 RPM
System Fan 3A     | 34h | ok  | 29.5  | 5890 RPM
System Fan 5A     | 38h | ok  | 29.9  | 5890 RPM
CPU 1 Fan         | 3Ch | ok  | 29.11 | 5820 RPM
CPU 2 Fan         | 3Dh | ok  | 29.12 | 5400 RPM
```

The sample command in the following example demonstrates use of raw commands to read fan energy sensors. To access lower-level capabilities in the BMC it may be necessary to provide unique one-byte value sequences to indicate a specific command and associated parameters. It is sometimes necessary to use raw commands with IPMItool because not all BMC commands are captured as individual command-line options. Instructions on how to construct raw commands are included in the IPMI specification, and instructions on how to specify associated parameters for the name and location of sensors are captured in documentation provided by the server manufacturer. The names and locations of sensors can vary by platform.

The following command returns energy for one of the fans. The last 8 bytes returned by the command include 4 bytes for running energy in millijoules and 4 bytes for running time in milliseconds. The command is executed twice to illustrate common usage. Periodic reading of these sensors allows users to calculate power. For example, subtracting the first command's energy from the second command's energy provides an energy delta. The energy delta between two commands can be divided by the time delta to calculate power.

```
# ipmitool -I lan -H xeon-bmc -U root -P pass -b 6 -t 0x2c raw 0x2E 0xFB
0x57 01 0x00  0x3 0x0

57 01 00 19 f5 13 01 c7 e3 17 00

# ipmitool -I lan -H xeon-bmc -U root -P pass -b 6 -t 0x2c raw 0x2E 0xFB
0x57 01 0x00  0x3 0x0

57 01 00 f5 54 14 01 fb eb 17 00
```

■ **Note** IPMItool can also be used to access Node Manager functionality since Node Manager is connected to the BMC using an IPMB link.

---

www.dbooks.org

More information on component-level power management can be found at www-ssl.intel.com/content/www/us/en/data-center/data-center-management/node-manager-general.html.

Operating System Monitoring Tools

Many times, users are interested in getting a high-level picture of what applications and the operating system are doing and are less interested in diving into the architectural and micro-architectural details. Commonly used tools for Linux (*SAR*) and Windows (*Perfmon* and *Logman*) provide an excellent first stop for information about the power and performance characteristics of software.

SAR

SAR is a Linux tool that monitors processor time, power states, scheduling, memory, I/O, and many other operating system visible events. SAR collects events over a user-defined time interval and outputs many event counts as per-second averages. For several of the monitored events, SAR provides additional detail below a system-level view. For example, processor time can be measured for each individual logical processor, and I/O statistics can be measured for each individual drive or network interface. SAR can be used to gain extensive insight into resource use.

The user-defined time interval and the number of intervals to use in data collection are defined by command-line parameters. For example, the following command specifies `-A` to measure all events, once per second, over 120 seconds. A full list of available command-line options is available by invoking the tool's help option, and additional information can be readily found online.

```
# sar -A 1 120 > sar.dat
```

The following shows a sample of the output and includes a portion that monitors context switches and interrupts. SAR measures interrupts both at the system level and per IRQ. Users can use SAR output along with `/proc/interrupts` and `/proc/irq/*/smp_affinity` to determine what specific devices are generating the interrupts, how frequent they are, and where they are being handled.

```
08:46:14 PM    proc/s   cswch/s
08:46:24 PM      2.24 104049.64

08:46:14 PM      INTR    intr/s
08:46:24 PM       sum 198321.77
08:46:24 PM        19      1.12
08:46:24 PM        99   7552.09
08:46:24 PM       100   7574.47
08:46:24 PM       101   7297.25
08:46:24 PM       102   7580.26
08:46:24 PM       103   7472.13
```

```
08:46:24 PM       104   7808.85
08:46:24 PM       105   7774.36
08:46:24 PM       114   7650.66
08:46:24 PM       115   1660.83
08:46:24 PM       116   1443.44
08:46:24 PM       117   1593.69
08:46:24 PM       118   1750.97
08:46:24 PM       119   1373.35
08:46:24 PM       120   1674.57
08:46:24 PM       123   1417.90
08:46:24 PM       124   1298.17
08:46:24 PM       125   1331.54
08:46:24 PM       126   1375.99
08:46:24 PM       127   1025.64
08:46:24 PM       128   1324.62
08:46:24 PM       129   1425.33
08:46:24 PM       130   1441.81
08:46:24 PM       131      0.71
08:46:24 PM       132      0.51
08:46:24 PM       133      0.51
08:46:24 PM       134      0.51
08:46:24 PM       135      0.51
```

Perfmon and Logman

Perfmon is a Windows tool that monitors processor time, power states, scheduling, memory, I/O, and many other events. Perfmon also allows applications to add their own events to the Perfmon infrastructure, allowing users to monitor performance from an application's perspective alongside the operating system events. Perfmon can be used to develop extensive insight into resource use.

Perfmon events can be visualized in real-time using the GUI (shown in Figure 7-18) or they can be collected for offline analysis using the Windows Logman tool. Logman provides command-line automation of Perfmon as well as other monitoring features such as event traces. It allows users to define different data collectors, or sets of monitoring events, and control when and how the data is collected. Users have the options of creating an always-running monitoring log, initiating different data collection scripts at different times, and writing output to multiple formats.

www.dbooks.org

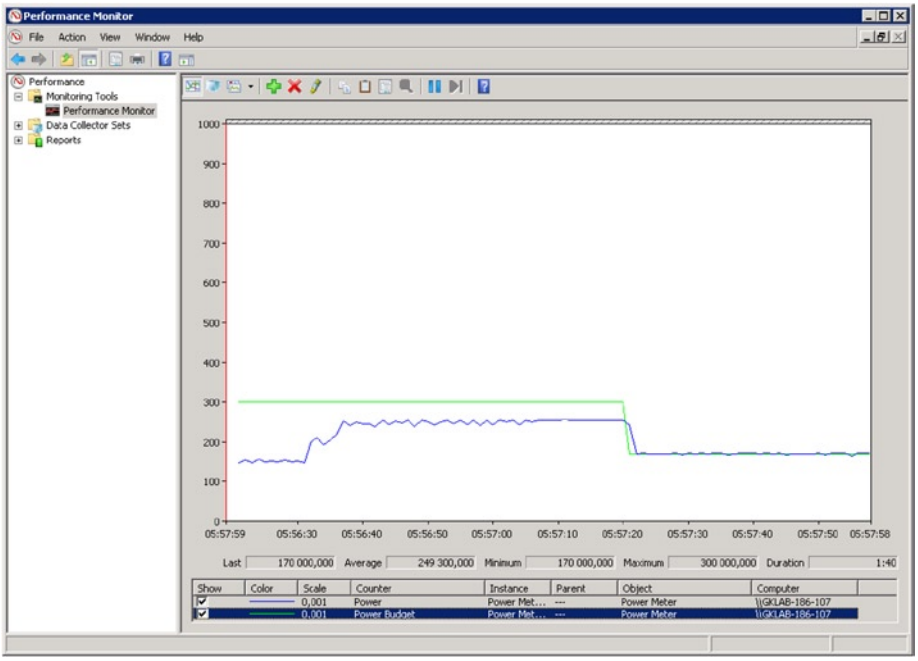


Figure 7-18. *The Node Manager ACPI Power Metering counters in Performance Monitor showing a power limit being enforced*

The Windows `typeperf` command will list all available Perfmon events on the system. Users can redirect output of this command to a file and edit the file as they see fit to include only the events of interest. The resulting file can then be used as input to Logman to define and create a new data collector. Here is an example of this command:

```
C:\Windows\System32\typeperf.exe -q > input_file
```

The following example command creates a new data collector called TEST. Logman includes many command-line options to define collection interval, output format, and output location. A full list of available command-line options is available by invoking the tool's help option.

```
C:\Windows\System32\logman create counter TEST
--v -ow -f csv -si 12 -rf 00:01:20 -cf \path \input_file -o \path
```

The following example command starts the TEST data collector. This will monitor the system for 120 seconds and write to a comma-separated value (CSV) file as defined by the command-line options used to define the data collector.

```
C:\Windows\System32\logman start TEST
```

The following output shows a sample of the output including a count of C2 and C1 requests per second issued by the operating system.

```
12/4/2013 12:53:46 PM        147563.4769         42.47840145
12/4/2013 12:53:58 PM        147825.0577        180.6790098
12/4/2013 12:54:10 PM        149276.3783        120.0697295
12/4/2013 12:54:22 PM        148175.1804         70.16651533
12/4/2013 12:54:34 PM        145440.724          83.24879264
12/4/2013 12:54:46 PM        144713.8251        145.7252465
12/4/2013 12:54:58 PM        146356.6638        112.1651317
12/4/2013 12:55:10 PM        147555.4464        102.7483379
12/4/2013 12:55:22 PM        147615.9768         46.81029614
12/4/2013 12:55:34 PM        148977.9331        106.331977
```

Summary

Numerous capabilities and tools exist for monitoring a system to understand power and performance characteristics. Different types of monitoring capabilities, including hardware monitoring, management controller monitoring, and software monitoring, have unique benefits and usages that aid in understanding system behavior.

Simple metrics can be used to convert raw monitoring data into formats more suitable for analysis, and a number of software tools can aid in visualization. Common monitoring tools, example use, and example output outlined at the end of this chapter provide a quick-start guide for monitoring. Chapter 8 will continue by discussing monitoring techniques that can be used to guide optimization decisions, along with specific examples of tuning.

www.dbooks.org

CHAPTER 8

Characterization and Optimization

Servers have a wide variety of different hardware and software configuration options. These include simple options such as enabling and disabling a feature. It also includes more advanced options that allow operators to control usage conditions, functionality, or other feature behavior. The individual features discussed in this book are primarily for power management—for example, P-states, C-states, link, and device states. Servers also have a large number of configurable features that are designed for performance including Turbo, memory prefetchers, or memory controller page policy.
In addition, servers have a large number of configurable features that are designed to add functionality, such as virtualization; security; or reliability, availability, and serviceability (RAS) features.

Servers have an equally large number of software options including different operating systems, applications, and management software options. For any given workload, a server's default hardware and software settings are designed to provide a good balance between low power and high performance. However, for many workloads, these default settings may not be optimal or may not be in line with an operator's performance, power, or cost goals. Most platforms enable power management features, so they are used aggressively when utilization is low. They are used conservatively as utilization increases, and they are disabled when the server reaches full capacity. At full capacity, power management features are either explicitly or implicitly disabled, so power management has no negative impact to maximum throughput.

All of the different types of features just described have an impact on performance, power, or other factors. The specific impact varies significantly based on the workload being measured and the system components being used. In many cases, the only way to definitively understand how various configuration options will impact power or performance is by experimenting on a specific system configuration. For example, an I/O-intensive workload with simple transactions may realize an undesirable performance impact from PCIe L1. However, the impact will vary significantly depending on the I/O (network or storage), device latency, device bandwidth, and whether the system is running at low or high utilization. There is no one-size-fits-all answer to questions about individual feature performance or power characteristics.

Before determining whether to tune, which features to tune, or how to tune, operators need to determine their requirements, such as functionality, power, and performance. It is also important to identify optimization goals. An operator who is optimizing for maximum performance may make decisions regardless of power consumption and will end up choosing a very different set of features and tunings than an operator who wants to minimize power. Other operators may take a more balanced approach; they may seek to lower power as much as possible, but at the same time, still meet a response time (performance) requirement. Other operators will use power management as aggressively as possible as long as those features do not impact maximum throughput. For a small number of servers, the investment in advanced feature tuning may not be worthwhile. However, the value increases significantly as the number of servers deployed increases.

■ **Note** The process outlined in this chapter frequently refers to tuning to decrease power or increase performance and the tradeoffs between the two. You can use the same process and analysis techniques to assess other tradeoffs, such as cost or functionality. In addition, requirements may extend beyond power and performance, such as temperature or reliability limits.

Feature tuning is very straightforward with the right tools and process. Chapters 2-6 describe various power management features, including how they work, and feature power and performance characteristics. Chapter 7 described how to monitor those features to understand their use. This chapter describes a process for characterizing and optimizing servers for the datacenter.

The following steps provide an overview of optimizing a server. Many of these individual steps will be described in greater detail throughout the remainder of the chapter.

1. Set power and performance requirements and optimization goals. For example, minimize energy while meeting a response time requirement or maximize performance below a temperature limit.
2. Collect data in the target environment to understand runtime characteristics. This includes data collected using power, performance, and thermal monitoring capabilities over a range of use conditions.
3. Analyze data to identify gaps relative to requirements and to understand what improvements the operator needs to make to reach optimization goals.
4. Analyze data to uncover new issues and opportunities. For example, an operator may identify during characterization that they only use a fraction of the available memory capacity or that the number of software threads running is frequently less than the number of logical processors.

www.dbooks.org

5. Create a test environment for tuning experimentation. Identify a workload that is representative of the target environment. Reuse industry and open source workloads that model similar applications and services or create new workloads based on the target environment's key characteristics.
6. Identify options to tune including BIOS setup options, OS options, and application options.
7. Measure the target workload and collect data with server default settings. This represents the baseline measurement that all future experiments will be compared against.
8. Measure the target workload with each identified feature and tuning, making only one change at a time.
9. For each change, collect and analyze data to identify the power and performance impact.
10. Identify those changes that aid in meeting requirements and optimization goals and measure the target workload with a combination of these.
11. Deploy beneficial changes to the target environment.
12. Repeat the process whenever there is a significant change to requirements, optimization goals, use conditions, or system components.

Workloads

Workloads are software services, applications, or testing tools that measure the performance of a server. They attempt to model representative usage scenarios based on usage conditions of interest. Workloads provide a repeatable way to measure performance of a system and are particularly useful when you are experimenting with and trying to understand the impact of changes to hardware or software in a datacenter.

Performance can be represented in a variety of different ways depending on the workload of interest. For example, throughput metrics, such as transactions per second or I/O per second, measure the peak performance capabilities of the platform. Latency metrics, such as transaction response times, time to completion for compute jobs, or I/O (drive and network) latency, measure the responsiveness of the platform. Power metrics, such as platform power, memory power, frequencies, and voltages, are used to understand the energy cost per unit of work. Several workloads bring together a combination of the metrics of interest, providing performance per watt or performance per dollar.

Workloads can be used to study a subset of system components, the system as a whole, or a cluster of servers. A profound understanding of system behavior can be gained when representative workloads are coupled with extensive data collection (such as core and uncore performance monitoring units, digital power meters, and OS metrics) and the results are carefully analyzed.

Identifying Suitable Workloads

The first requirement for any workload is that it is measurable. A service, application, or testing tool must have one or more metrics captured during a measurement that can be used to convey various measured throughput, latencies, or a composite metric of the two.

Workloads must be repeatable. If an experiment is conducted twice, three times, or a hundred times without any configuration changes in-between, the measured results must be the same. Workloads with poor repeatability make it difficult or impossible for operators to tell whether a measured change in performance is due to a configuration change or simply due to normal experiment variation. Ideally, workloads used for experimentation have less than 1% variation in measured power and performance; however, 2%-3% is common. If variation exceeds 5%, it presents a problem because the workload can no longer be used to assess the impact of smaller or more subtle changes to system configuration.

Workloads must be reproducible—executing the same transactions or computations, using the same inputs, and following the same order or distribution for every measurement. Reproducible workloads measure performance during a timed interval that is the same for every measurement. They also can be reset to a starting state that is identical across measurements. For example, workloads that utilize a database must be able to restore a backup database before every measurement.

Workloads or systems with poor scalability can affect how repeatable or reproducible a measurement is. For example, a workload that is being used to measure compute performance will not shed light on changes if there is an I/O bottleneck. Similarly, a workload that only utilizes a few threads may not accurately illustrate the performance difference between an eight-core or an eighteen-core processor.

Workloads must be representative. Representative workloads perform similar transactions or computations as the scenario being modeled and also use the same software stack and configuration as the scenario being modeled. For example, a representative Infrastructure as a Service (IaaS) workload would utilize a virtualized environment. It would have virtual machines utilizing heterogeneous applications, it would vary the load on the server over time, and it would vary the number of running instances. Many workloads include random elements to improve their representativeness—for example, workloads that vary the client think time or the interarrival rate of transactions.

Workloads that are not representative typically only model a small portion of the scenario of interest. This makes it more likely that the tuning results from a test environment will not apply to the production environment. For example, a testing tool that measures single-threaded TCP roundtrip latency wouldn't be representative of a web server. An operator could make several changes to improve the performance of the test workload that would have no impact or a negative impact on their production workload.

Another key attribute for characterization and optimization is whether the workload is configurable; for example, does the operator have the ability to change the problem size in a scientific workload or to change the number of connected clients in a transactional workload? Having ample configuration options is key to tailoring the workload setting to best match an environment of interest.

www.dbooks.org

Workload Types

There are a wide variety of different workload types—for example, testing tools, energy efficiency benchmarks, industry benchmarks, and datacenter workloads. Each has different applications, different purposes or goals, and a different level of complexity.

Testing Tools

Testing tools don't necessarily model a representative service or application; instead, they are used to stress a single component or subset of related components in the system. For example, Intel provides a testing tool to characterize cache and memory performance. The Intel Memory Latency Checker tool (Intel MLC) can be used to measure maximum memory bandwidth, idle latency, and loaded latency.[1] Although these tools are great for testing some key system characteristics, it is also relatively easy to misinterpret the results. One common mistake is to measure idle memory latency with clock-enabled (CKE) power savings enabled. These power savings commonly engage during idle latency tests, causing a significant increase in the idle latency. CKE tends to have much smaller impacts on real system performance.

There are also testing tools that focus on I/O. The open source iperf workload is popular for measuring network bandwidth; the NetPIPE workload is popular for network latency. For storage, the open source FIO or IOmeter tools are popular and flexible and allow users to vary different I/O parameters and think times.

Testing tools can give a preliminary indication of how a given feature might impact power or performance. They are also very helpful in identifying the base capabilities of a platform and can be used as guides for detecting bottlenecks. Individual components can be monitored when testing tools are being measured to understand state residencies, bandwidth, or throughput limits. Chapter 7 provides an outline of the different types of metrics to look at when monitoring components.

It can be easy to misinterpret the results of some micro benchmarks. For example, idle latency benchmarks (such as the Intel MLC) will exhibit significant increases in latency as a result of memory CKE power savings features (see Chapter 3). Not only will CKE result in an increase in idle latency due to the CKE wakeup, but the precharge powerdown (PPD) feature will also result in closed memory pages (and further latency increase). It is not uncommon for users to draw the conclusion that CKE causes significant performance loss based on this benchmark. In practice on real workloads, CKE has a much smaller impact on latency.

Energy Efficiency Workloads

Energy efficiency workloads are used to study both the power and performance characteristics of a system. They exercise CPU and memory and provide a great preliminary analysis of system behavior. The most popular energy efficiency workloads are SPECpower_ssj2008 (SPECpower) and the Server Efficiency Rating Tool (SERT) from

[1]See https://software.intel.com/en-us/articles/intelr-memory-latency-checker.

the Standard Performance Evaluation Corporation (SPEC). SPECpower is widely used across the industry and features several years of published results, so there is a great amount of data to use for system comparisons.

SPECpower

SPECpower measures simple transactions running in a Java Virtual Machine (JVM) across a broad range of CPU and memory utilization. It includes idle, maximum throughput, and a range of load points in between those endpoints. It provides a way to visualize power consumption at various ratios of maximum performance. This visualization, with power on one axis and performance on the other axis, is commonly called the *load line*. There are examples of this visualization spread throughout this book; however, many examples use a workload other than SPECpower. The load line methodology introduced by SPEC in SPECpower has seen broad use across the industry because it can be easily applied to different workloads.

Figure 8-1 shows CPU power during a SPECpower measurement. It illustrates several calibration load points used to determine system performance, as well as several load points of varying utilization all the way down to idle.

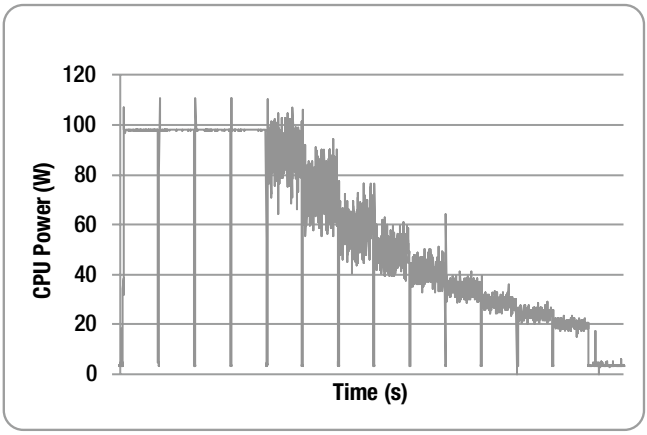


Figure 8-1. *CPU power measured over various load levels of SPECpower*

SPECpower has some representative characteristics, but it lacks more sophisticated transactions that are common in datacenter workloads. It does not exercise a complete software stack and does not have any significant storage or network I/O. As a result, SPECpower includes some characteristics that are not commonly seen in production workloads. SPECpower has an order of magnitude fewer power state transitions compared to typical datacenter workloads. Residency in low-power states is much higher in SPECpower than in other workloads. In SPECpower, transactions are initiated in batches, rather than being individually initiated by network connected clients. This means the workload is both idle and active for longer periods of time, leading to very different P-state behavior than is seen with a typical datacenter workload.

www.dbooks.org

Due to the workload's focus on CPU and memory, certain features or platform-level changes may improve SPECpower scores, but they do not provide a benefit in a typical datacenter workload. As is the case with every workload, there are optimizations that may benefit SPECpower that aren't generally a good idea to apply elsewhere.

■ **Note** A significant challenge in workload selection is balancing ease of use with representativeness. Workloads that model realistic use scenarios tend to include a large number of network-connected clients, sophisticated software stacks, and multiple different systems under test. Workloads that are easy to install, measure, and maintain do not.

The Server Efficiency Rating Tool (SERT)

SERT is a new SPEC tool suite under development that measures power and performance across a broader range of use conditions and applications. Rather than executing a single test, it includes a variety of different worklets that stress the CPU (different types of operations, including mixing integer, floating point, data references, and modification), memory, and drives separately. SPECpower loads are one of the worklets, so SERT extends the type of analysis that can be done above and beyond SPECpower.

SERT has been adopted by the Environmental Protection Agency (EPA) for their Energy Star program for servers. SERT is also being investigated for use in energy efficiency programs by government agencies around the globe including Europe and Asia. It is interesting to note that SERT is being used not only for power and performance assessments, but is also being considered for government-based environmental programs.

Industry Workloads

Industry workloads are typically two- or three-tier workloads with transactions driven over the network based on more realistic transaction interarrival rates. These workloads use representative application and software stacks and often include quality of service or response time requirements. The ability to measure transaction response time is a key capability because it allows operators to characterize the performance impact of power management features. The downside to using industry workloads is that they represent a significant resource investment, both in engineer time and hardware.

Industry workloads are one of the best tools available to characterize and optimize a server. These workloads are maintained and updated by various open source efforts and an industry consortium, and they see extensive use across the industry. The majority of charts in this book were generated using monitoring power and performance with industry workloads.

A number of good industry workloads span various market segments. For example, many HPC and scientific workloads span life sciences, computer-aided engineering, financial modeling, and weather simulation, and many of these are open source. Similarly, there are a number of enterprise workloads modeling database and mail servers, customer relationship management (CRM) systems, and enterprise

resource planning (ERP) systems, and there are also many cloud workloads that model multitenant environments or that model environments with distributed services that utilize the Web, memory cache, and databases.

The Transaction Processing Performance Council (TPC) and SPEC are two industry organizations that develop and maintain workloads. TPC workloads such as TPC-C (order-entry OLTP), TPC-E (brokerage-firm OLTP), and TPC-DS (decision support system) are popular for power and performance characterization. SPEC workloads such as SPECweb (web server), SPECvirt (infrastructure consolidation), and SPECimap (mail server) provide similar capabilities. In addition to the industry organizations, there are also a number of company-sponsored workloads such as SAP Sales and Distribution (SAP SD) workload or VMware VMmark.

In addition to the workloads maintained by industry organizations, a number of good open source workloads span various market segments: Olio (web, social networking), mcblaster (object cache), HammerDB (OLTP), and TPoX (Transaction Processing over XML), for example.

■ **Note** Open source workloads are an excellent starting point for workload development or for customizing a workload to better model a target environment.

Industry workloads are more complex and realistic, and they represent a step above. Unlike SPECpower, most industry workloads do not have integrated power and performance metrics, and they do not automatically generate a load line. However, the load line concept from SPECpower can be easily applied to industry workloads.

Industry workloads allow users to create a variety of different loads, typically specified by a number of virtual clients or delay time between client transactions. For example, Olio requires an operator to specify the number of users that will be used to generate different types of web transactions. Measurement across a variety of different utilization or throughput levels is possible by running with a different number of users. For many virtual workloads, several thousand virtual clients are required to reach maximum throughput, so it is easy to make fine-grained adjustments to load based on varying the number of users.

Idle Workloads

As is the case with industry workloads, the conditions used to test an idle server should also be representative. Many idle power measurements are taken shortly after a server is powered on and initialization is complete. The server may not have any applications or services initialized and ready for use. This state is called *operating system idle*. In operating system idle, it is typical to see long, uninterrupted idle durations and excellent residency in package C-states.

To get a representative measurement, it is best to measure idle on a system that has all applications and services initialized and has recently completed running a representative workload. This state is called *active idle*, and unlike operating system idle, it may include intermittent network activity, periodic management and security

www.dbooks.org

operations, and sporadic application activity. Active idle for virtualized environments typically includes a variety of different virtual machines running different software stacks, adding to its complexity. As the name implies, active idle is significantly more active than operating system idle, leading to higher power. Figure 8-2 illustrates the difference in activity between operating system idle and active idle with numerous applications and services loaded.

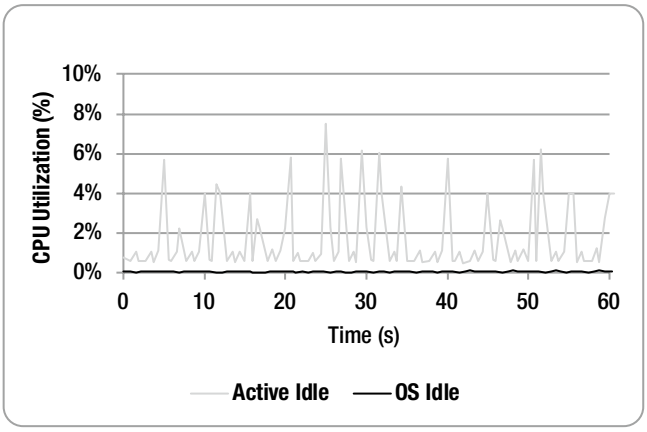


Figure 8-2. *Variation in CPU utilization between active idle and OS idle*

Measuring and comparing the differences in operating system idle and active idle can help to identify software and hardware components that impact idle power. For example, applications that utilize polling rather than events or applications that change the timer frequency of the operating system can both result in higher idle power. Because most servers spend a significant amount of time idle, optimizing for idle can be as beneficial as optimizing for active loads. The tools operators can use to diagnose active idle issues are discussed in Chapter 7.

System Characterization

Hardware and software monitoring tools are key capabilities needed for system characterization. The best techniques for data collection can change based on the workload type or based on the amount or type of data collected.

Steady State vs. Non-Steady State

When monitoring workloads it's important to identify whether the workload of interest is steady state or non-steady state. During a steady state workload, system characteristics don't change significantly over time. Industry benchmarks that model transactional systems are typically steady state workloads; TPC-E or SPECweb are examples of this type of workload. These workloads execute a predetermined mix of transactions, repeatedly, over a very long period of time.

For many workloads, there is a significant amount of ramp-up time before steady state is reached. When a workload first starts, application and system characteristics are changing frequently as clients connect, load is balanced, and frequently-used data is cached.

Non-steady state workloads consist of frequently changing system characteristics. It's common for application behavior to change from one second to the next—for example, a scientific application modeling weather patterns or a data analytics workload with different map and reduce phases. Figure 8-3 shows how the rate of instruction retirement varies for a steady state and non-steady state workload. The multiple different phases of the non-steady state workload can be clearly identified.

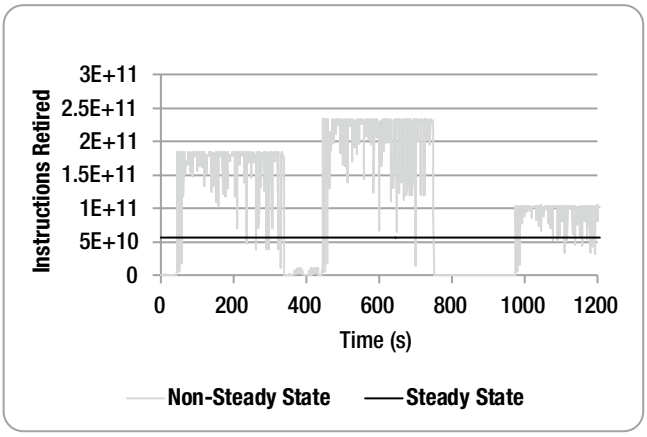


Figure 8-3. *Variation in instructions retired between non-steady state and steady state workloads*

Data Collection

There are a number of important considerations for collecting data during workloads. If data collection tools are executed too frequently or multiple tools are executed in parallel, it can alter the behavior of the workload. If data collection is not started and stopped at a consistent point in time, data cannot be compared between measurements.

Collection Duration

The start time and length of data collection depends on whether the workload is steady state or non-steady state. For steady state workloads, power and performance data is typically not collected during ramp-up since initialization phases are not of interest. Similarly, data collection can be skipped during ramp-down phases or when a workload has finished and is restoring files to a known starting state.

When data is collected during the steady state between ramp-up and ramp-down, it is only necessary to collect data for a small portion of time because system characteristics don't change significantly from one moment to the next. Data collected

www.dbooks.org

during one minute of steady state should have characteristics very similar to data collected during the next minute of data collection. Data collection may need to be started and stopped for workloads that have several significant intervals. For example, SPECpower may require tools to be started and stopped at specific times to allow for load points to be individually analyzed.

For non-steady state workloads, performance and power data is typically collected throughout the entire duration of a measurement. Unlike steady state workloads, it is not reasonable to collect data for a smaller portion of time since the characteristics of data collected during one minute of time can be very different from the data collected during the remainder of the workload.

Collection Frequency

If power and performance events are collected at high frequency, data collection tools will interrupt applications frequently, stealing CPU cycles from the workload of interest. This may alter the natural behavior of the workload, revealing power or performance issues that would not occur when data collection is stopped. If events are collected at low frequency, interesting workload characteristics may be difficult to discern. Operators will be unable to both identify unique phases of the workload and determine how power and performance characteristics change over time.

■ **Note** Unlike hardware monitoring tools, operating system tools can collect hundreds of different events in a single interval. Collecting data at low frequency can still perturb the workload if too many events are collected at the same time.

For steady state workloads, data collected at low frequency will produce a similar result as data collected at high frequency. This simplifies the data collection process for steady state workloads since the workload can be characterized with only a small amount of data. For non-steady state workloads, high frequency collection is necessary. Finding the ideal data collection frequency will require some experimentation as it varies from workload to workload. The goal is to collect data as frequently as possible while having no impact on system power or performance.

Event Ordering and Event Groups

Many times it is necessary to analyze several events collected at the same time to get a complete picture of component or system behavior. For example, to characterize memory references during a particular phase of a workload, an operator may want to measure read transactions, write transactions, page hits, page empty accesses, page misses, and memory latency. As discussed in Chapter 7, this may not be possible since many monitoring units can only collect a small number of events in parallel.

For steady state workloads, this isn't a significant issue. Monitored events measure very similar values from one moment to the next, so it is possible to gain the desired insight into component or system behavior by splitting up events over several different groups measured at different times. For non-steady state workloads, the limitation of monitoring units can be an issue. The best practice is to collect related events as close (in time) to each other as possible and to collect the same event groups repeatedly at high frequency.

Multiple Tools

During workload characterization, operators may be required to measure data using several different tools targeting monitoring capabilities spread across software, firmware, and hardware. When multiple tools are used, there are some cases where it is beneficial to measure multiple tools simultaneously. For example, to provide complete coverage for a non-steady state workloads or when there is interest in comparing software metrics to hardware metrics collected at the same time. There are other cases where it is beneficial to measure multiple tools one at a time. For example, to limit how intrusive data collection is during a steady state workload.

Similar to the process of determining collection frequency, event groups, and event ordering, there are several choices for how to collect data using multiple tools. These require some experimentation as each of those choices has unique strengths and weaknesses.

Methodology

Characterizing and optimizing a server requires not only good workloads and a good data collection strategy, it also requires good methodology. The following recommendations serve as guidelines for basic server characterization.

- Quality test each server before investing time in characterization and analysis. Use testing tools to confirm the configuration is healthy, and measure the target workload several times to ensure workload variability is within acceptable limits.
- Always collect data with monitoring tools. Without proper visibility into power and performance characteristics, the result of changes to software or hardware from one measurement to the next will not be well understood.
- Always use a consistent baseline measurement, making no change to the configuration throughout the duration of an experiment. For example, if a driver or BIOS is updated, drive capacity is added, or a network is reconfigured, measurements on the server can no longer be compared to previous measurements.
- Only change one variable at a time between measurements. For example, coupling several software optimizations together in a measurement may result in no change in power, whereas in reality, some of the optimizations may be helpful while the others are harmful.

www.dbooks.org

- Always automate workloads and data collection. This ensures measurements will collect data with the same start time, end time, collection frequency, and event ordering every time. Without this consistency in timing, comparisons between different measurements may provide misleading results.
- Use a controlled thermal environment. Repeatability of a workload can be impacted if the temperature varies significantly based on weather, time of day, or activity of other systems.

Analysis

With thousands of different events that can be monitored during a measurement, using a top-down analysis is one of the best strategies for analyzing changes to a server. First, it is important to look at the big picture. The most important metrics to analyze are power, performance, and cost since most feature tuning will result in an increase or decrease in one or more of these. Performance will be measured in throughput and latency (response time or time to completion) and will be collected by the target workload. Power will be measured either using a digital power meter or through the power supply or current sensors using the monitoring features described in Chapter 7.

Power Metrics

Configuration and tuning changes can cause very large differences in system power. Figure 8-4 highlights how substantial the change can be. The lowest power tuning in this example is able to minimize the energy cost per transaction without impacting maximum throughput.

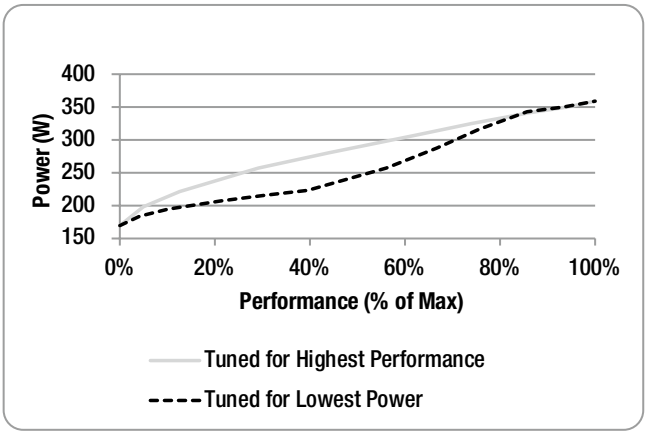


Figure 8-4. *Comparison between power and performance (throughput)—Server OLTP Workload*

To build a deeper understanding of a configuration or tuning change, first identify the component or components that caused the change. Then, identify specifically why those components' power changed. For a given power increase or decrease, there are several events and metrics in the data collected that can be used to identify the cause.

In an ideal scenario, the system is capable of component-level power measurements, and the components contributing to a change in power can be identified simply by comparing the power of individual components (processors, memory, PCH, LAN adapters, drives, and fans).

However, systems with component-level power measurement capability are uncommon today. A more assessable approach to determining a change in power is to identify differences in active and idle states and operating conditions. These are the primary factors that ultimately determine power consumption. For example, a change in CPU power could be identified by analyzing CPU C-state and P-state residency and transitions. A change in fan power could be identified by analyzing each fan's tachometer, and a change in interconnect power could be identified by analyzing interconnect voltage, frequency, and link width.

If power measurement and monitoring capabilities are insufficient to identify what caused a change in power, performance and thermal events can be used to provide alternative insight. For example, if a measurement showed a 10-times increase in IOPS to a SSD, this increased activity may indicate a measurable increase in drive power.

A change in system power is always the result of a single component change because there are many complex dependencies between various components in the system. For example, an operator could decrease memory power by limiting the memory frequency used and capacity installed; however, this change might increase system power as CPUs are stalled waiting for data to be returned from memory and drives. As the workloads used become more sophisticated, the number and complexity of component and system dependencies increase. For example, tuning any given node in a cluster of servers running a distributed application may result in a small local difference but a substantial global difference when the cluster is viewed as a whole.

The following list serves as a prioritized guide for top-down power analysis. Although this list is targeted for power analysis, performance metrics are also valuable, and will be covered momentarily. See Chapter 7 for more details on how to monitor these various metrics.

- CPU Power
    - P-state residency
    - C-state residency and transitions
    - Temperature
- Memory power
    - Frequency and voltage (static at runtime)
    - Self-refresh and CKE residency and transitions

www.dbooks.org

- Thermal management
  - Platform temperatures
  - Fan power
  - Tachometer
- Device power
  - Link width and frequency
  - I/O bandwidth and transactions per second
  - QPI L1 and L0p residency and transitions
- Other
  - Additional system performance metrics

Performance Metrics

Configuration and tuning changes can cause very large differences in system performance. Figure 8-5 highlights how substantial the change can be. This figure underlines the importance of performance requirements. If this server had a performance requirement that 95% of transactions are completed in less than 10 milliseconds, then the lowest power tuning would decrease power and cost while still meeting that performance requirement. However, if the server had a performance requirement that 95% of transactions are completed in less than 2 milliseconds, an operator may both tune for performance and load systems to no more than 80% of maximum capacity.

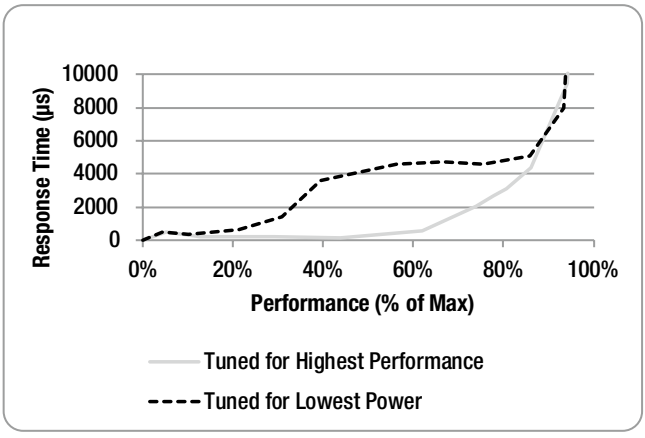


Figure 8-5. *Comparison between response time and performance (throughput)*

To build a deeper understanding of a configuration or tuning change, first identify the component or components that caused the change. Then identify specifically why those components' performance changed. This process of performance analysis is easier if the target workload's sensitivity to CPU frequency and I/O subsystem performance is well understood.

■ **Note** One good method for determining sensitivity to CPU performance is through frequency scaling studies. This involves running a target workload several times across a range of fixed frequencies. The results allow operators to calculate scaling efficiency between two frequencies by comparing the percent increase in performance to the percent increase in frequency.

If a workload exhibits poor frequency scaling, transaction time is likely dominated by waiting for memory or I/O. Alternatively, the poor frequency scaling may be the result of a bottleneck. For example, a datacenter workload that drives line rate network traffic is unlikely to see a significant increase or decrease in performance based on CPU frequency.

Differences in performance can be identified by analyzing the changes in CPI, path length, or power state residencies. For example, a decrease in operating frequency will almost certainly result in a decrease in performance. An increase in CPI could be the result of a degrading cache hit rate or increasing memory latency. The source of an increase in path length can be identified by analyzing execution profiles collected by tools such as Linux perf. The execution profile will allow an operator to drill down to the specific process, modules, or function that is exhibiting a change in behavior.

The following list is a prioritized guide for top-down analysis. Note that system power, covered in the previous section, is very important to consider when understanding changes in performance. For example, monitoring C-state residency and transitions not only tells the operator about time spent in state, it also tells about accumulated exit latency (C0 impact) and effects of flushing caches (CPI impact). See Chapter 7 for more details on how to monitor these various metrics.

- CPU performance
    - Cycles per instruction (CPI)
    - Cache misses and latency
    - Memory latency
    - C-state transitions (latency) and residency (C0)
    - Path length
    - Software execution profiles
    - P-state residency (frequency)
    - Thermal throttling

www.dbooks.org

- Memory performance
  - Memory latency
  - Self-refresh and CKE residency and transitions
  - Memory bandwidth
  - Thermal throttling
- Interconnect power
  - QPI L1 and L0p residency and transitions
- Device performance
  - Link width and frequency
  - I/O bandwidth and transactions per second

Optimization

There are a large number of different optimization opportunities in the system, and it can be challenging to know which are worth the effort. This section explores a variety of different optimization opportunities and investigates the potential tradeoffs involved. In addition to highlighting valuable opportunities, it will cover a selection of items that you may want to avoid. Some optimizations must be performed at boot in the BIOS, whereas others may be possible at runtime. This chapter provides recipes, where possible, for how to make different changes in the system.

Power management algorithms can also provide improved performance in select cases. This section will highlight some of these opportunities for improved performance.

CPU Power Management

Typically, the CPU is one of the first places to start performing power (and performance) optimizations because it frequently contributes significantly to the overall power in the platform. However, there are times when the CPU is not as significant to the overall power consumption of the platform. A good example is a storage system with a large JBOD (just a bunch of disks). In such a system, the CPU power may not be a major contributor to the overall power, and it may not be worth the effort to focus on CPU optimizations. One of the first tasks in performing any optimization work is to determine where the power is going. Total platform power should be characterized and compared against the CPU subcomponent power (see Chapter 7 for details). It is common for the CPU (and memory) to contribute a significant percentage of the overall platform power, but this should be confirmed prior to focusing significant effort on optimizations.

Chapter 2 provides background on many of the features discussed in this section.

P-States and Turbo

Voltage and frequency can play a significant factor in power consumption. Running at a lower frequency when performance is not required can save significant platform power, particularly on high TDP Xeon processors.

Running at lower frequency (P-states) will likely increase the response time of workloads when they are running at low to moderate utilizations. However, as shown in Figure 8-5, many workloads exhibit much higher overall response times when running at peak utilization (with or without power management) than what is observed when power management is enabled at lower utilizations. As a result, the response time impact of these features may not be the dominant component in the worst-case latency situation.

Turbo can be used to increase the achieved frequency beyond the base frequency. On recent processor generations, it has been common for Turbo to provide a 10%–20% (or more) peak performance increase. A common misconception is that Turbo provides "burst" performance for only a short period of time. Although this is frequently true in thermally constrained consumer devices, it is generally not the case in server deployments. Server workloads can frequently sustain some level of Turbo indefinitely.

RAPL (and other frequency management algorithms) can engage while Turbo is running to limit the frequency of the system. Frequency is typically managed by these algorithms on (small) millisecond granularities. Frequency transitions block execution for about 10 to 20 microseconds. The algorithms have been tuned so that these transition periods have minimal impact on the overall throughput (and performance) of the system. However, users who are very sensitive to latency disturbances, such as high-frequency traders, may not want to use Turbo in order to avoid execution being blocked. Many server workloads are able to easily tolerate this latency cost, and the benefits provided by the increased frequency far outweigh the latency cost.

A common misconception is that Turbo frequencies are less power efficient than running with Turbo disabled. Although this is true on some processor SKUs, it is not always the case. Some lower power and lower frequency products achieve optimal platform performance per watt while running in Turbo.

Modern operating systems take tens of milliseconds to detect changes in demand and utilization. As a result, the use of OS-controlled P-states can result in short periods of time where the operating frequency is lower than what would be best for the demand of the system. One of the potential upsides of hardware power management in future products is the ability to improve the response time to changes in demand and utilization.

■ **Note** The use of P-states and Turbo does not have to be a yes-or-no question. Modern operating systems can be constrained to request frequencies within a range. This can result in significant power efficiency savings with contained impacts to response time.

P-states and Turbo need not be an all-or-nothing decision. For example, a user has a system that typically runs at ~60% utilization, but periodically it has an increase in demand. That user's system has a SKU that can run at an "all-core Turbo" (P0*n*) frequency of 3 GHz and a base frequency (P1) of 2.6 GHz. When the customer runs the system with Turbo requested all the time, there is a notable increase in peak performance

www.dbooks.org

(which is useful for those periods of high demand), but it comes at a notable power increase during the typical levels of demand. Running with all of Turbo and P-states enabled results in good power efficiency, but at times, it has undesirable response time characteristics. In such a case, the user could instruct the OS to always request at least 2.4 GHz. Under such a configuration, the user is able to avoid the power cost of running at 3 GHz at typical utilizations, but then transition up to 3 GHz when demand increases.

■ **Note** It is recommended that both Turbo and P-states be controlled through the operating system and not through the BIOS. This provides significant flexibility for a longer term to changes in decisions without requiring system reboots (which can be very undesirable in large-scale deployments).

The BIOS does have the ability to disable Turbo, and many BIOS designers expose this option to end users. However, there is no way to disable frequency transitions in the system. Some BIOSes include an option to disable EIST (Enhanced Intel Speedstep, but this will only change how P-states are enumerated to the OS in ACPI.

■ **Note** Some OEMs have proprietary mechanisms for managing frequency that exist between the OS and the CPU by leveraging capabilities made possible by ACPI. Special care may be necessary if these capabilities are enabled in the OEM's BIOS.

Frequency Control in Windows

In Windows, frequency can be managed through the Power Options control panel as well as by using the `powercfg` command line utility. The High Performance tuning option (available both in the control panel and through `powercfg`) will instruct the OS to request the maximum frequency at all times. By default, the Balanced mode will request frequencies across the spectrum, including Turbo if it was enabled by the BIOS.[2] Under the Advanced Settings options, you can find additional tuning options to control the range of frequencies that are selected (see the Minimum Processor State and Maximum Processor State configuration options). Note that these options do not control actions in the Turbo range.

The `powercfg` utility can also be used to manage both frequency selection and Turbo.[3] Turbo can be disabled with the following command line (or enabled by changing the 0 to a 1):

```
powercfg -setacvalueindex scheme_current sub_processor PERFBOOSTMODE 0
powercfg -setactive scheme_current
```

[2]Note that Intel processors prior to the Westmere generation had Turbo disabled by default in the Windows Balanced configuration mode.

[3]See the *Performance Tuning Guidelines* for more details about `powercfg`. https://msdn.microsoft.com/en-us/library/windows/hardware/dn529134.

Frequency selection outside the Turbo range can also be controlled with `powercfg`:

```
powercfg -setacvalueindex scheme_current sub_processor PROCTHROTTLEMIN 60
powercfg -setacvalueindex scheme_current sub_processor PROCTHROTTLEMAX 100
powercfg -setactive scheme_current
```

The current system configuration can be dumped from `powercfg` with this command:

```
powercfg -Q,
```

Frequency Control in Linux

The Intel P-state Linux driver provides a mechanism for constraining the requested frequencies in `sysfs`.[4] These are located today in

```
/sys/devices/system/cpu/intel_pstate/
```

The max_perf_pct, min_perf_pct, and no_turbo files can be used to configure the desired P-state behavior on a per-logical-processor granularity. The percentage values that are configured here include control in the Turbo range, making it possible to limit the maximum frequency used even in the Turbo range. Although it is not possible to determine which part of the range is for Turbo today using the `sysfs` interface, you can disable Turbo at runtime by executing the following (as root):

```
# echo 1 >> /sys/devices/system/cpu/intel_pstate/no_turbo
```

The algorithms that select frequency can be further tuned using `debugfs`. However, most users have generally reported minimal benefit by moving away from the default configurations.

Prior to the introduction of the `intel_pstate` driver, the `acpi_freq` driver was used by default to manage frequency in Linux. Little effort has been spent in recent years to optimize this driver to work well in a server environment, and it is not recommended. One drawback of the `intel_pstate` driver is that support is required in the actual kernel for it. It is not possible to load the driver as a module on an older kernel. Note that it is possible to fall back to the older `acpi_freq` driver on kernels that include support for `intel_pstates` using the boot-time kernel parameter `intel_pstate=disable.` ,

The Linux kernel also supports the concept of P-state "governors," which control the aggressiveness of the frequency selection algorithm. With `acpi_idle`, the `ondemand` and `performance` governors were common in server deployments. The `performance` governor simply requested the max performance at all times, whereas `ondemand` attempted to change the frequency based on system utilization. Many users experienced performance issues with the `ondemand` governor, pushing them to use the `performance` governor (which effectively disabled P-states outside of Turbo). With `intel_pstates`,

[4]See www.kernel.org/doc/Documentation/cpu-freq/intel-pstate.txt for details.

www.dbooks.org

ondemand is no longer an option. Instead, intel_pstates provides the performance and powersave options. The performance governor operates like it used to—the system will request the max frequency at all times. The powersave governor replaces ondemand and provides a new algorithm (compared to acpi_freq), which is intended to provide power savings without some of the performance drawbacks that were observed with acpi_freq ondemand.

Turbo Ratio Limit

As described in Chapter 2, processors generally have varied maximum Turbo frequencies based on the number of active cores. These limits are fused into each unit and are fixed across a given processor SKU. Users can, at runtime, reduce the maximum allowed Turbo frequencies based on the number of active cores using MSRs 0x1AD, 0x1AE, and 0x1AF. The exact semantics of thee MSRs is processor specific (and summarized in Figure 8-6).

Cores Active	Sandy Bridge		Ivy Bridge		Haswell	
	MSR	Bits	MSR	Bits	MSR	Bits
1	1AD	[7:0]	1AD	[7:0]	1AD	[7:0]
2	1AD	[15:8]	1AD	[15:8]	1AD	[15:8]
3	1AD	[23:16]	1AD	[23:16]	1AD	[23:16]
4	1AD	[31:24]	1AD	[31:24]	1AD	[31:24]
5	1AD	[39:32]	1AD	[39:32]	1AD	[39:32]
6	1AD	[47:40]	1AD	[47:40]	1AD	[47:40]
7	1AD	[55:48]	1AD	[55:48]	1AD	[55:48]
8	1AD	[63:56]	1AD	[63:56]	1AD	[63:56]
9	-	-	1AE	[7:0]	1AE	[7:0]
10	-	-	1AE	[15:8]	1AE	[15:8]
11	-	-	1AE	[23:16]	1AE	[23:16]
12	-	-	1AE	[31:24]	1AE	[31:24]
13	-	-	1AE	[39:32]	1AE	[39:32]
14	-	-	1AE	[47:40]	1AE	[47:40]
15	-	-	1AE	[55:48]	1AE	[55:48]
16	-	-	-	-	1AE	[63:56]
17	-	-	-	-	1AF	[7:0]
18	-	-	-	-	1AF	[15:8]
Semaphore	-	-	1AE	[63]	1AF	[63]

Figure 8-6. *Turbo ratio limit configuration*

Starting with the Sandy Bridge generation, specific Turbo frequencies could be requested directly by the operating system. As a result, controlling Turbo frequencies with these MSRs may not be necessary in all conditions. As an example, if you are simply looking to set a maximum requested frequency with intel_pstates, this can be done with sysfs (as described previously). However, in other OSs where ACPI is used, there may not be a user interface to the OS for this level of control. In such situations, these MSRs can be used.

These MSRs can also be used for fine tuning the Turbo levels based on the number of active cores. However, in practice, such configuration is generally not needed. On Sandy Bridge servers, MSR 0x1AD was sufficient to control the limits. Each byte in the register specified the ratio for a given core count. There was a maximum of 8 cores, so the 64-bit MSR was sufficient.

Ivy Bridge servers support up to 15 cores. MSR 0x1AE was added in order to provide 1 Byte per core. The user was required to first configure MSR 0x1AD and then 0x1AE. When writing to MSR 0x1AE, bit [63] needed to be set to 1 to instruct the hardware to take the configuration.

Haswell servers supported up to 18 cores. MSR 0x1AF was added, and the semaphore bit was moved out to MSR 0x1AF [63].

Note that these MSRs are package-scoped, meaning that one copy is shared across all cores on a given socket. When writing these registers, you should generally perform the writes from one logical processor on each socket in the system.

Turbo Ratio Limit provides a mechanism for users to find a "compromise" between enabling and disabling Turbo. For example, a user may find that using all of Turbo results in undesirable frequency transitions, but disabling Turbo significantly reduces peak performance and throughput. By reducing all maximum Turbo ratios to some level in between P1 and P0*n*, the user may be able to find a sustainable Turbo frequency (on a given workload) that does not generate any frequency transitions. This can be a useful compromise between completely disabling Turbo and using the full Turbo capabilities.

On Haswell processors, the use of AVX instructions could reduce the maximum Turbo frequencies. The user could use Turbo Ratio Limits to set the max Turbo frequencies to match the levels achieved with AVX, providing more consistent frequency as workloads transitioned between AVX sections.

Note that due to scalability concerns, this approach may be discontinued on future processor generations and replaced with an adapted interface.

Uncore Frequency Scaling

Uncore Frequency Scaling (UFS) was introduced on Haswell E5/E7. This feature autonomously controls the frequency of the uncore based on a variety of metrics inside the CPU. UFS not only saves power, but also works to share power with the cores in order to provide higher frequency and improved performance. UFS transitions block all system traffic for about 20 μs today. As a result, these transitions have been tuned to be rare (milliseconds between transitions).

The UFS frequency algorithm can be controlled using the non-architectural MSR 0x620. [7:0] controls the minimum ratio; [15:8] controls the max. With these two fields, the algorithm can be bounded. The internals of the algorithm cannot be configured and may change from generation to generation.

The base (untuned) UFS algorithm has been tuned for performance on most enterprise workloads. A common (and simple) alternative configuration is to set this MSR to 0x3F3F, which will attempt to lock the uncore at the highest frequency allowed by the processor SKU. This configuration is useful for latency-sensitive usage models. Many networking users have also found this configuration to be desirable for their usage models. Note that the TDP frequency assumptions assume that UFS is allowed to be dynamically managed, and locking the frequency at the maximum may result in frequency reduction below P1 on very high-power workloads. In practice, this is not observed on real workloads.

www.dbooks.org

Core C-States

In addition to P-states and frequency controls, Core C-states provide some of the biggest impact CPU power across a range of system utilizations. As described in Chapter 2, it is important to remember that Core C6 can provide a significant performance boost when paired with Turbo by allowing periods of time where only a few threads are active to operate at higher frequencies. This can be valuable in certain parallel workloads where Amdahl's Law[5] is at work.

Unlike P-states, which slow down the rate that instructions are executed, C-states provide power savings at the cost of a "wake-up" when execution is resumed. These wake latencies are typically in the tens of microseconds for C6 and about a microsecond for C1. Just like P-states, this will ultimately manifest itself as an increase in response time.

The core will enter the C1 state when the `HLT` (halt) instruction is executed. This is part of the instruction set and C1 cannot be disabled in hardware. Core C1e can similarly not be directly disabled by hardware. However, there is a configuration bit that the BIOS can set that causes C1 requests to be promoted into C1e requests. This is frequently documented as "C1e Enable," but that definition is not strictly true. C1e on Windows is generally enabled and disabled using this bit as a result of the way that ACPI enumerates C-states to the operating system. This is also the case with older versions of Linux using the ACPI idle driver. The Linux `intel_idle` driver will automatically disabled the promotion of C1 to C1e independent of BIOS configuration so that it can autonomously select between those two states based on the system behavior (C1 is used when a short idle period is predicted, whereas C1e will be used for slightly longer idle periods).

Core C6 is used when `MWAIT(C6)` is executed by the operating system. There is no hardware disable for Core C6 on current processors. The BIOS has the option of selecting which C-states are enumerated to the operating system using ACPI. Different BIOS designers expose this to the customer in different manners. Disabling C6 through the BIOS will effectively disable it on both Windows and older versions of Linux using the `acpi_idle` driver.

Linux with the `intel_idle` driver today will not look at the BIOS configuration when deciding what C-states to use. Because `intel_idle` does not look at ACPI, it is not possible for the BIOS to communicate this information to the driver. Instead, it uses the `intel_idle.max_cstates` kernel parameter to control the level of C-states used. This definition is product- (and even kernel-) specific, so some experimentation may be required. Note that setting this to a value of 0x0 will disable the driver rather than disable C-states, so values greater than 0x0 are recommended

There is also a demotion algorithm that can autonomously decide to use a shallower C-state than what the OS has selected. In general, it is not recommended that users manipulate this configuration.

■ **Note** Today the CPU will not autonomously grant a deeper C-state than the one that the operating system requested.

---

[5]The speed-up of parallel computing is limited by the percentage of a workload that is run in a serial manner.

Runtime Core Disable

C-states can provide a performance boost by providing access to higher Turbo frequencies and also by saving power in a power-constrained environment. Although C-states will autonomously be requested by the operating system if no work is pending for that core, it is also possible to provide a hint to the OS that a given CPU should not be used. This will prevent the OS from scheduling tasks to that core so that it can stay in a deep C-state. Note that this behavior is not 100% robust, because cores can still be woken up in certain cases (e.g., a broadcast thermal interrupt). However, in practice, they tend to be very effective.

On Linux, this is called core offline and can be done on an individual core basis using `sysfs`.

```
#echo 0 > /sys/devices/system/cpu/cpuX/online
```

On Linux, the logical processors to physical core mapping (including sockets and hyperthreads) is technically controlled by the BIOS. The topology can be discovered using `sysfs`:

```
/sys/devices/system/cpu/cpu*/topology/*
```

Although it is recommended that the topology be discovered and decoded, it is useful to understand what to expect in general. In practice, the mapping is mostly consistent across generations and configurations and is best described with an example (see Figure 8-7 for an illustration). Consider a two-socket system with two four-core processors that each support HT. The logical processors that are enumerated in `sysfs` will be enumerated as follows:

- Logical processors 0 to 3 represent the first thread on cores 0 to 3 on socket 0.
- Logical processors 4 to 7 represent the first thread on cores 0 to 3 on socket 1.
- Logical processors 8 to 11 represent the second thread on cores 0 to 3 on socket 0.
- Logical processors 12 to 15 represent the second thread on cores 0 to 3 on socket 1.

www.dbooks.org

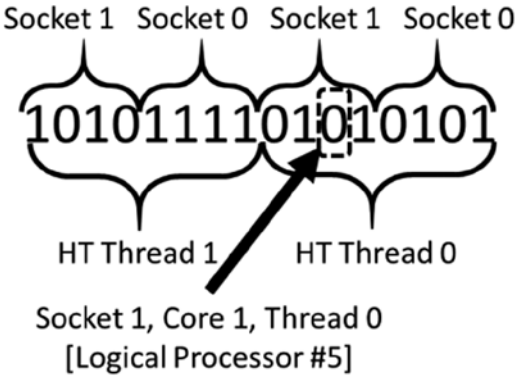


Figure 8-7. *Example logical processor bitmask with two-socket, four-core CPUs on Linux*

Windows has a related feature called C*ore* P*arking*. This feature is a bit different from the offlining support that is available in Linux, and it does not provide a mechanism to force certain cores to stay turned off. Core Parking is no longer recommended in server deployments by Microsoft[6] and has been disabled by default.

Although Windows does not have a way to force specific logical processors to turn off across the operating system, specific applications can be affinitized to a set of logical processors. In conjunction with OS requested C-states, the user can use affinity control to encourage the OS to not use specific cores, allowing them to enter into a deep C-state. The logical processor to physical core mapping can be decoded using the `coreinfo`[7] utility. Affinity control is beyond the scope of this book.

Package C-States

Unlike Core C-states, which are under the control of the operating system, package C-states today are autonomously managed by the CPU. See Chapter 2 for more details on package C-States. Package C-states save additional power when all of the cores are in a deep C-state by taking additional power savings steps that would not be possible if cores were active. These states are generally only possible at very low system utilizations, and their residencies can be monitored with software, as described in Chapter 7.

One of the biggest impacts of package C-states is that they increase the latency for external devices to communicate with DDR memory. In order to access memory, the CPU must wake from the package state, which typically takes tens of microseconds.

When it uses package C-states, the idle power of the Xeon E5 CPU is on the order of 10-15 W. With Core C-states enabled and package C-states disabled, the CPU power will increase by a moderate amount (the amount is dependent on the processor generation). Note that this power increase will be amplified by power delivery efficiency losses (see Chapter 4), since power delivery tends to be less efficient at lower power levels. A rough rule of thumb is a ~1.5–2 times increase.

[6]See `http://support.microsoft.com/kb/2814791`.
[7]See `https://technet.microsoft.com/en-us/sysinternals/cc835722.aspx`.

Package C-states can also impact platform power. This impact can be even larger than what is observed on the CPU die. The most notable impact comes from memory self-refresh. If opportunistic self-refresh is disabled, then disabling package C-states will result in memory only using CKE to save power. This can have a large impact on platform idle power savings, particularly on systems with a large memory capacity.

As described earlier in this chapter in the "Idle Workloads" section, one limitation with package C-states is that it is common for the software that continuously runs on them to result in an "active idle" state that is not truly idle. These states are most effective when the system is able to achieve a truly idle state across all sockets in the node. This is different from many consumer usage models, which are aggressively optimized and tuned for idle power due to battery life concerns. In order to take full advantage of idle power savings opportunities, additional work may be required to tune (or stop) software that prevents the system from becoming truly idle.

If a user has a system that spends a notable amount of time at idle, then enabling package C-States can save a notable amount of system power. This is particularly true of systems with large memory capacities. Package C-states are typically controlled by the BIOS. The OS does not have direct control over these states today. It is also possible to disable them by writing 0x0 into MSR 0xE2 [2:0] (CST_CONFIG_CONTROL).

Energy Performance Bias

A number of power management algorithms manage different power and performance tradeoffs in the system. Some of these algorithms are internal and their tunings are not externally visible. Energy Performance Bias (EPB) provides a mechanism for software to provide a hint to the CPU about how the user would like the system to make these tradeoffs. Today, this is an architectural MSR that provides 4 bits to select from up to 16 different operating modes. Smaller values represent more performance whereas larger values save more power. Rather than requiring users to attempt to tune 10 different features that could have different tunings on each project, this feature is intended to provide a simple interface for tuning CPU power management algorithms.

On Xeon E5/E7 processors starting with Sandy Bridge, EPB supported four modes of operation and the two least significant bits of the MSR were ignored. In practice, only two modes have ultimately shown significant value: a Performance mode (EPB values 0 to 3) and a Balanced Performance mode (EPB values 0 to 7). Two deeper modes (Balanced Energy and Energy Efficient) exist, but in practice, it has been difficult to distinguish them from the Balanced Performance mode. Table 8-1 provides an overview of some of the key features that are managed by EPB.

Table 8-1. *EPB Feature Control Summary*

Feature	Performance	Balanced Performance
Turbo demotion	Disabled	Enabled
Memory CKE	Disabled	Enabled
QPI L0p	Disabled	Enabled
Internal algorithms	Performance tunings	Power savings

www.dbooks.org

Dynamic switching is another feature that was introduced on Sandy Bridge E5 that automatically transitions the system into the Performance mode when the CPU detects that high performance is desired. It is recommended that this remain enabled. Some users may desire to override some of these decisions that are typically controlled by EPB and Dynamic EPB Switching, and these limited opportunities will be discussed later in this chapter. Note that dynamic switching will not occur on systems that have disabled Turbo, so selecting the right EPB mode is more important on such systems. Users who configure the system to request Turbo 100% of the time (such as the Windows or Linux `performance` governors) will also switch the system into Performance EPB mode 100% of the time.)

EPB has not been optimized on Atom servers or on Xeon E3 servers like it has been on the E5/E7 product lines. The dynamic switching algorithm is also only productized on E5/E7.

When it was initially architected, EPB was placed in control of the operating system with 0x1B0 IA32_ENERGY_PERF_BIAS. On Windows, the Power Options control panel Performance mode will automatically select the EPB Performance mode. On Linux, the easiest option is to simply write directly to the MSR on all threads. EPB is not managed dynamically on Linux.

On Sandy Bridge, a configuration bit was added so that the BIOS could take control of EPB away from the operating system. This is controlled by the non-architectural MSR 0x1FC (MSR_POWER_CTL). If bit [25] is 0, then the EPB is controlled by the OS and MSR 0x1B0. Otherwise, the OEM is in control. In these cases, EPB may be controlled through some existing BIOS option to further simplify the process for end users.

One of the biggest impacts of EPB is the Turbo demotion algorithm. Turbo demotion has no impact for customers that have disabled Turbo. In such a situation, using the EPB Performance mode may be the best choice. However, such a configuration will result in CKE being disabled, which may be undesirable. Overrides are available for CKE to ensure that it will be used. This will be discussed later in the chapter.

For customers that are making use of Turbo, the Balanced Performance mode has been shown to effectively save power with minimal impact to performance on most workloads due to dynamic switching. One drawback to this situation occurs when there is a spike in system demand. Similar to the operating system, it will take time (tens to hundreds of milliseconds) to detect the spike in demand and react accordingly.

Note that recent versions of the Linux kernel will detect an EPB boot configuration of 0x0 (Performance) and reprogram it automatically to 0x6 (Balanced Performance). As such, you may want to configure this mode using the operating system instead of the BIOS.)

Hyperthreading

Hyperthreading (HT) allows two logical processors to share a single core. By sharing a core, they frequently reduce their thread performance in order to achieve higher overall throughput and higher system performance. The act of enabling HT has almost zero impact on power by itself. However, by having multiple threads share the resources of a core and increase the overall performance and throughput of the core, power is frequently increased. Running a core with HT enabled, but having the scheduler only use one of the two logical processors on that core, will result in an almost identical power consumption to disabling HT.

Although HT will frequently increase the power consumption of a core, it almost always does so in a power efficient manner. It is typically one of the most power efficient performance optimizations that exists; however, some workloads that do not benefit from HT do exist. Workloads that are very sensitive to memory bandwidth are a good example. Such workloads may see a slight efficiency loss by executing them with HT. However, in practice, disabling HT in order to improve power efficiency is very rarely an optimization that pays dividends.

HT can frequently be disabled in the BIOS. It is also possible to instruct Linux not to use HT threads using the same mechanism as core offlining. This provides *effectively* the same behavior as the BIOS disable. The `/affinity` flag can be used in Windows for a similar effect.

Prefetchers

Prefetchers can provide significant performance boosts on certain types of workloads. On others, they provide little value. Some workloads even lose performance with prefetchers, but this has become much less common over the years as memory bandwidth has improved and the state of the art has improved with prefetcher design.

Prefetchers almost always increase the amount of memory bandwidth being consumed by the system. Even workloads that see no performance benefit from prefetching may observe a measurable increase in memory bandwidth. This memory bandwidth will increase platform power. As a result, users may see power efficiency improvements by disabling prefetchers on workloads that exhibit little or no performance upside.

MSR 0x1A4 can be used to configure prefetchers on many server processors. Bits [3:0] are generally mapped to different types of prefetchers in the system. By setting these bits to a 1, the corresponding prefetcher(s) are disabled. Setting all four bits to 0xF is an effective way to disable prefetching. This can be performed at runtime in order to enable easy testing or deployment. Some BIOSes also provide an interface to disable prefetchers and use the same interface. MSR 0x1A4 is not an architectural MSR and therefore may change in the future.

PCIe

The primary power saving feature for PCIe is L1. L1 enabling has generally struggled in the server ecosystem over the years. It saves a relatively small amount of power (on the order of ~1 W for an x8 connection on the CPU, with additional savings on the connected device) at the expense of increased latency for PCIe devices that make use of the feature.

On some Haswell E5 platforms, PCIe L1 has a larger impact on idle power than previously observed on Sandy Bridge and Ivy Bridge. On some platforms, additional platform-level power optimizations depend on all links being in the L1 state. This can change L1 from a 1 W feature into a 10 W feature. Users should experiment with L1 to best understand the tradeoffs on a given system.

ASPM L1 enabling is controlled by the BIOS. However, the operating system can also be used to disable ASPM. Enabling ASPM from the OS if it was not enabled by the BIOS is generally not recommended.

www.dbooks.org

In Linux, ASPM can be disabled using the `pcie_aspm=off` kernel parameter. ASPM can also be disabled using `sysfs`:

```
/sys/modules/pcie_aspm/parameters/policy
```

One can ascertain the current ASPM configuration using the following:

```
> lspci -vvvv | grep ASPM
```

In Windows, you can determine if ASPM is enabled or not with `powercfg /energy` report. ASPM can be configured with `powercfg` (assuming it was enabled by the BIOS). The following will disable ASPM:

```
powercfg -setacvalueindex scheme_current sub_pciexpress aspm 0
```

Customers who are concerned about PCIe latencies should investigate L1 disable. Others who are more power conscious should evaluate ASPM L1, particularly if their deployments are already making use of package C-states.

QPI

QPI supports two main power management features: L1 and L0p (as described in Chapter 3). QPI L1 is only utilized during package C-states and has zero cost due to the fact that the L1 wake-up is done in parallel with other longer latency operations. L0p is used at runtime and does result in some power and performance tradeoffs.

Starting with Ivy Bridge E5/E7, L0p was disabled automatically by dynamic switching or when the EPB performance mode was selected. L0p increases cross-socket data movement latencies by ~10 ns. This can result in a small decrease in performance (up to ~1%). Users that have disabled Turbo and desire maximum performance should consider setting EPB in performance mode.

QPI can be configured to run at lower frequencies. The QPI voltage is fixed and the power savings from running at lower frequencies tends to be relatively small. Bandwidth across the link is directly proportional to the QPI frequency, and there is also a small latency cost (<10 ns) when QPI is run at low frequencies. In practice, the power savings from running QPI at lower frequencies is not worth the performance and flexibility impact.

On some systems (particularly two-socket platforms), multiple links exist between sockets. It is possible to disable links in order to save notable power (on the order of ~10 W at low, but non-idle utilizations) at the expense of a significant decrease in cross-socket communication bandwidth. This can lead to severe performance loss on some workloads. However, a subset of workloads, such as those that primarily execute out of the cache or have extremely good NUMA optimizations and locality, may observe minimal performance loss. By monitoring the QPI link bandwidth, you may be able to determine whether this could be a good opportunity for power savings. It is not possible to enable or disable links at runtime.

Memory

Memory can be a large contributor to overall platform power in deployments with large memory capacities. Memory naturally has a wide power dynamic range, because a large percentage of the power is a function of the memory bandwidth (independent of power management actions like CKE and self-refresh). A good rough estimate of memory power is a simple linear function where both the slope and y-intercept are a function of the type of memory deployed (capacity, ranks, process generation, etc.). Memory power management and the power characteristics of different types of memory technologies are discussed in Chapter 3. In addition to the inherent power scaling that exists in memory, additional CPU-driven power management capabilities can be tuned and configured to provide different power and performance characteristics.

As a general rule, users with small memory capacities likely need not get overly aggressive with memory power management, whereas those who are loading up the DIMM slots should carefully consider their options here. "Small" is a difficult term to formally define in this case. With the transition to DDR4, power consumption has significantly improved, making aggressive power management somewhat less important. On low-power servers with TDPs that are less than about 50 W, the contribution of memory power is proportionally larger, and a quick look at the memory power management configuration is advisable. For higher power servers, users who are deploying x4 devices in 2DPC configurations (or larger) should evaluate their options. Users with small deployments, such as a 1DPC x8 population, may want to look at other opportunities for power savings first.

CKE

CKE provides moderate levels of power savings with minimal performance penalty. As a result, it is commonly used while applications are active to save power during short idle periods. Using CKE while performance benchmarking generally results in a peak performance decrease of ~1% and a minimal impact on response times because it only costs ~10 ns to wake up memory from this state. Memory latency benchmarks suffer from CKE though, particularly because the PPD variety forces a page close. This latency increase is not representative of the performance impact on real workloads.

On systems that support dynamic switching and EPB (such as Xeon E5/E7 class servers starting with Sandy Bridge), CKE is disabled automatically in the Performance mode (or after a dynamic switch). Using EPB Balanced Performance with dynamic switching enabled is effective at avoiding the 1% peak throughput decrease while saving power at lower system utilizations for systems that have Turbo enabled and have not configured their OS to request max frequency 100% of the time. Users who configure their system to use EPB Performance mode or request maximum frequency 100% of the time should consider enabling CKE at all times if they have a large memory topology and are willing to trade off ~1% peak throughput. This configuration must be done through the BIOS, and different BIOS OEMs may make such a configuration available through different mechanisms.

www.dbooks.org

Some platforms have provided options for both APD and PPD modes.[8] In practice, the difference between these two modes of operation is not a first order impact on either power or performance. APD, for example, sounds like it should have slightly better performance characteristics (at slightly higher power), but in practice, it is largely not significant. As a result, tuning these modes is only recommended for users who are looking to fine-tune their system to a specific workload, and only after they exhaust other tuning opportunities.

On Avoton/Rangeley, EPB and dynamic switching do not control CKE. It must be statically configured by the BIOS. Due to the low SoC TDP power of these systems, smaller memory capacities will have a larger contribution to platform power, and users should consider their options with respect to CKE. One big exception here is with storage deployments that have a large number of HDDs. In this case, the memory power is frequently dwarfed by the drive power, making CKE less important overall.

Self-Refresh

Self-refresh provides much larger power savings than CKE at the cost of significant wake-up latencies (generally on the order of a few microseconds). As a result, it is typically targeted at large idle periods, such as when all cores are asleep. Aggressive use of self-refresh can result in performance loss and even an energy increase for completing a set of work.

Conceptually, there are two ways to use self-refresh. First, it can be used during deep package C-states. In this situation, the latency cost is effectively zero, since the wake-up can be done in parallel with other actions. We will refer to this as *ForceSR* for simplicity. Secondly, it can be used outside of package C-states. This is referred to as *Opportunistic Self-Refresh (OSR)*. Because self-refresh is typically "free" during package C-states, controls for this capability are not made available for the user. OSR configuration is separate from the Package State configuration.

Unlike CKE, OSR (and Force SR) is not automatically disabled by EPB Performance or dynamic switching. This decision must be made in the BIOS. By default, OSR is configured to be very unaggressive. It is intended to target very long idle periods where package C-states are not active, such as in a system with very good NUMA optimizations or when package C-states have been disabled. It can also be useful in saving power in multi-socket systems at idle that are running software that prevents high package C-states residency but does not actually frequently access memory on all channels/sockets during the spurious events that are preventing package C-states. Tuning the aggressiveness of the algorithm is possible through the BIOS, but this is not recommended.

Recall from Chapter 3 that the CK behavior is an option for how deep of a self-refresh to use. The clocks can stay active, which significantly shortens the wake latency but also reduces the power savings effectiveness. In practice, self-refresh with the CK active has minimal power savings benefits over CKE. As a result, this is the option that is less interesting than the Enable/Disable decision.

In practice, for most users, the base OSR configuration is effective at saving power without an observable impact to responsiveness or throughput. However, latency-sensitive users who are concerned about block times on the order of ~5-10 μs should disable OSR in addition to package C-states.

[8]Haswell E5/E7 only productized the APD state.

Patrol Scrub

Patrol Scrub is a reliability feature that steps through memory-reading each line and writing it back. The intention is to detect correctable errors (and correct them) before they can degrade into uncorrectable errors. The bandwidth cost of Patrol Scrub is quite small and does not materially impact the power of the system at runtime. However, Patrol Scrub does prevent OSR. Only a subset of channels on a socket is scrubbed at a time, though, so OSR is still possible on the remaining channels. Patrol Scrub is a very effective feature for avoiding uncorrectable errors, and it is not recommended that it be disabled. If a user observes that certain channels just won't enter OSR, the likely explanation is Patrol Scrub. Package C-states have been optimized to provide self-refresh (across all channels) while also maintaining a good average scrub rate.

NIC

Chapter 4 discusses some of the network interface card (NIC) power management options. Table 8-2 provides an overview of these capabilities. NICs can be connected into any system, and therefore the power management configuration is managed by the driver and not by the BIOS. This configuration can typically be managed at runtime.

Table 8-2. *NIC Optimization Summary*

Feature	Potential Savings	Cost	Description
Media Speed	Up to a few watts	Significant throughput loss and latency increase potential	The speed of NIC links can sometimes be reconfigured to save power at the expense of bandwidth. This can be done at runtime by software drivers, but it takes significant time to do so (during which time the network connection is blocked) and therefore it cannot be performed aggressively.
Energy Efficient Ethernet	~400 mW to ~2 W	<~16 µs	Idle power management feature for the network.
Interrupt Moderation	Platform dependent	Configurable, typically ~100-200 µs	Rate limits delivery of interrupts to the CPU, frequently resulting in power savings, improved throughput, or both, at the expense of latency.

www.dbooks.org

Interrupt moderation is one of the most interesting features for the NIC. It is common for users to concern themselves with features like P-states, which can induce latency bubbles on the order of 10 microseconds, while ignoring interrupt moderation, which can cause 10 times larger latency increases. Although this feature does increase latency, it is effective at both saving power and improving network throughput. Latency-sensitive users should consider changing the latency tuning parameter or even disabling this feature, but they should be aware that this will increase the demand on the CPU.)

DMA coalescing is another NIC feature. This feature has shown minimal effectiveness in server deployments. It is not enabled by default and is not generally recommended in servers.

Storage

Storage power management is made up of two components: the storage controller PHY and the device (HDD/SSD). These power management capabilities are standardized across both server storage subsystems as well as those found in consumer devices, and are discussed in Chapter 4.

As described in Chapter 4, SATA devices support four power savings modes: Working, Idle, Standby, and Sleep. Both Standby and Sleep can take significant time (seconds) to wake, and therefore they may not be desirable for server deployments. The Idle state, on the other hand, has minimal latency costs, and therefore it can be left enabled. For storage deployments where long latency wake-ups are acceptable, allowing the Standby state, or explicitly using the Sleep state, can help achieve very low idle power. Both SATA and SAS drives can enter a power management state either autonomously (after a configurable timer expires) or based on a command from the host. Table 8-3 provides an overview of this control. Remember that with HDDs, power consumption is heavily dependent on the use of power management actions, whereas with SSDs, power will automatically scale with bandwidth consumption.

Table 8-3. *SATA/SAS Drive Power Management Configuration*

Type	Host Request	Timer Configuration
SATA	Set Features: Go To Power Condition	Set Features: Extended Power Condition
SAS	Start/Stop Unit (SSU) command	Power Condition Mode page

In Linux, the `sdparm` and `hdparm` tools can be used to control these options. The `sdparn --all` command is useful for discovering if the drive supports mode pages for power management. Not all drives support the full set of configuration options, and the `–enumerate` flag is useful for exploring common mode pages (independent of a specific drive). For `hdparm`, the `-B`, `-S`, and `-M` flags can be used to control the power management aggressiveness of some drives. Other flags exist for requesting a drive to enter into a low-power state.

Smartmontools is another set of utilities that can be used to manage drives.[9] `smartctl` is one frequently-used utility that is part of this package. These utilities are commonly available on Linux but can also be used on Windows.

In Windows, the idle time before spinning down a drive can be set with `powercfg` (setting it to a value of 0 will disable putting the drive to sleep):

```
powercfg /Change disk-timeout-ac 15
```

The SATA Aggressive Link Power Management (ALPM) power state of *Partial* is able to achieve low idle power (~100 mW) with a wake-up latency of <10 µs. The deeper states have minimal power savings benefits in most servers and can cause significant latency. Users who are very latency sensitive can consider disabling all these states, but Partial provides a good compromise between latency and power savings.

In Linux, the tlp and tuned-adm tools are available (depending on the distro) for managing a wide range of power management options, including link power management.[10]

■ **Note** The tlp and tuned-adm utilities are useful for managing a variety of power management options in Linux.

In Windows, the link power management is controlled by `powercfg` in the *disk* subgroup. These options are hidden (not named) today. You can explore the current configuration using the following command. By doing this, you can ascertain the GUID associated with the hidden features and configure it in a manner similar to some of the previous command lines.

```
powercfg -qh scheme_current sub_disk
```

Thermal Management

CPU thermal management is configured to protect the system and is not tunable by consumers. Platform thermal management is managed be proprietary OEM algorithms, and therefore the specifics are beyond the scope of this book. OEMs commonly make different thermal management options available. More aggressive thermal management algorithms result in higher system temperatures and the potential for brief thermal throttling events and slight reductions in Turbo performance. In practice, these algorithms can save significant platform power without materially impacting the performance of the system.

[9] See www.smartmontools.org.
[10] See http://linrunner.de/en/tlp/tlp.html.

www.dbooks.org

Cooling a processor to very low temperatures tends to be cost prohibitive and unnecessary. Lower temperatures save leakage power and thereby improve Turbo performance of some workloads, but this effect is exponential and the benefits decrease rapidly as temperatures drops. The decrease in CPU leakage power typically is much smaller than the increase in cooling power.

Optimization at a Glance

This chapter has discussed a wide variety of different optimization opportunities, and it can be daunting to determine where to begin. What features should be enabled? Which should be disabled? Different users have different constraints and goals for what they are trying to achieve. This section provides a high-level summary of the various features that have been discussed. Before the optimizations are discussed, Table 8-4 provides a summary of some of the different types of impacts that these optimizations can have on the system. Tables 8-5 through 8-8 summarize a selection of the optimization opportunities, including the priority in which users may want to focus their efforts first.

Table 8-4. *Performance Metrics*

Metric	Description
Response time	Average time to complete a request from an external agent. Users of transaction processing workloads should note these.
Peak throughput	The feature may reduce (or improve) the peak throughput of the system. This could also be called “peak performance.”
Execution block	This feature may result in short periods of time (microseconds, unless noted otherwise) in which core execution may be blocked.
Device block	This feature may result in short periods of time (microseconds, unless noted otherwise) in which an external device may be blocked from access to memory.

Table 8-5. *CPU Optimization Summary*

Feature	Power Impact	Primary Performance Impacts	Utilization Targets	Control	Priority
P-states	Tens of watts	Response time,execution block,device block	Low to moderate	OS, BIOS	High
Turbo	Tens of atts	Peak throughput, execution block, device block	High	OS, BIOS	High
UFS	Tens of watts	Latency blocks, execution block, device block	Focus on low utilizations	BIOS, MSR	Medium
Core C-states	Tens of watts	Response time, execution block	Low to moderate	BIOS, OS	High
Package C-states	Watts	Response time, execution block	Idle	BIOS, MSR	High
Core Disable	Watts	Peak throughput	All	BIOS, OS	Medium
EPB	Tens of watts	Response time, peak throughput	Moderate to high	BIOS, OS	High
HT	Function of performance change	Peak throughput	High	BIOS, OS	Medium
Prefetchers	Workload dependent	Peak throughput	All	BIOS, MSR	Medium
QPI frequency	Minimal	Peak throughput	All	BIOS	Low
QPI link disable	Watts	Peak throughput	All	BIOS	Low
PCIe ASPM L1	Watts	Response time, device block	Low	BIOS, OS	Medium

www.dbooks.org

On many systems, the CPU is a significant component of the overall platform power. Table 8-5 provides some highlights from the CPU power optimizations that have been discussed. Note that the power impact shown here is for a typical high-TDP Xeon E5/E7 system. Not all systems will exhibit these impacts; the details are discussed in earlier sections.

Systems with large memory capacities or low-power CPUs may have large contributions for memory power. Table 8-6 provides a summary of some of the optimizations available for memory power savings. Users who are very latency sensitive will want to disable OSR, but typical users likely will not be exposed to it.

Table 8-6. *Memory Optimization Summary*

Feature	Primary Performance Impact	Utilization Targets	Control	Priority
CKE Enable	Peak throughput	All	BIOS	Medium
OSR	Response time, execution block, device block	Idle	BIOS	Low

Networking cards themselves do not contribute a large percentage of platform power in most systems. However, Interrupt Moderation is a key feature that provides tradeoffs between CPU utilization, power consumption, and latency. Tuning this feature to suit a user's needs can have a significant impact on the power/performance/latency characteristics of a system.) Table 8-7 provides a summary of some of the key NIC optimizations.

Table 8-7. *NIC Optimization Summary*

Feature	Power Impact	Primary Performance Impacts	Utilization Targets	Control	Priority
Media speed	Watts	Peak throughput, response time	All	Driver	Low
EET	Watts	Response time, device block	Idle	Driver	Low
Interrupt moderation	Watts to tens of watts	Peak throughput, response time, device block	All	Driver	High

In compute servers with only one or two drives, storage power is generally not a large contributor to overall platform power. In storage nodes, on the other hand, HDD and/or SDD power can dominate the power consumption of the platform. Table 8-8 provides a summary of the storage power optimization opportunities for a storage node. For compute nodes, it may be desirable to disable or "turn down" many of these features. Some can add considerable latency with minimal power savings. The Slumber state, for example, saves minimal power in a server environment but can result in millisecond wake latencies.

Table 8-8. *Storage Optimization Summary*

Feature	Power Impact (per drive)	Primary Performance Impacts	Utilization Targets	Control	Priority
PHY power state (Partial/Slumber)	Hundreds of mW	Response time, execution block, peak throughput	Idle	Driver	Medium
Device power savings	Watts	Response time, execution block, peak throughput	Idle	Driver	Medium

Summary

Power optimization can have a significant impact on both power consumption and performance in a platform. Users should start by characterizing their system behavior and power consumption so that they can decide which areas to focus on. There is no need to spend weeks attempting to optimize drive power if it is only consuming 5% of the overall platform power. Next, users need to identify a repeatable workload that can be used to best understand the power and performance tradeoffs of different optimizations. Once this is complete, users can identify targeted experiments on specific configuration changes based on the guidance provided in this chapter, and they can then identify the optimal configuration based on the their constraints and goals.

Finally, there are a few key things to remember when performing optimizations:

- P-states need not be an all-or-nothing decision. Enabling a small to moderate frequency range can save significant power with minimal exposure to performance problems. These decisions are best made in the OS and not with the BIOS.
- Higher frequencies can, in many cases, provide better platform power efficiency.
- C-states can provide significant performance improvements when paired with Turbo in addition to saving power.
- Interrupt moderation tunings can have a significant impact on response time, power efficiency, and throughput. Improved throughput and better power efficiency can be traded for faster response times.

www.dbooks.org

CHAPTER 9

Data Center Management

In prior chapters we discussed the optimization of the computing infrastructure inside the data center for energy efficiency. However, within the framework of the entire data center, this is only part of the energy story.

Data Center Management and Power Distribution

As mentioned in Chapter 1, the infrastructure surrounding computers in the data center is equally important to consider in the context of overall energy efficiency. If the infrastructure is required to support the computing, it needs to be included in the overall energy equation. Data center infrastructure itself has multiple missions; along with sheltering the computers from natural elements like humidity, extreme temperatures, and natural disasters, it provides office space for the engineers and technicians who operate the data center, and it manages the computing resources. The data center infrastructure handles energy delivery to the computing resources and the disposal of waste heat from them. It also fulfills a mission of resiliency by providing both physical security and some form of survivability planning in the event of power outages.

In this chapter, we will touch on many of these aspects, especially as they pertain to energy management in the data center.

Data Center Facilities

Data center facilities vary widely in form, scale, and architecture, depending on local conditions, economics, and data center requirements. For instance, data centers can be housed in large purpose-built structures, as special purpose spaces within existing buildings, or in previously existing buildings adapted for a new purpose.

Large purpose-built data centers—such as those built by large Internet companies like Google, Apple, Facebook, and Microsoft—tend to be located in geographies that provide low-cost power and have close proximity to large populations centers (with proximity generally measured by ping times of less than 10–20 milliseconds) they serve. These data centers may be built with facility powers ranging from approximately 1 to 20 megawatts.

By way of contrast, many special purpose and general purpose data centers are housed within existing buildings. These data centers tend to be smaller and tend to be built to serve local or specialized purposes. Although smaller data centers can be built to high efficiency energy efficiency standards, in many cases these are of secondary importance to other considerations such as security, proximity to a specific physical location, or simply convenience.

Among the most interesting facilities are those built in buildings either renovated or adapted from another purpose. For instance, Google recently built a data center inside a converted paper mill in Finland.[1] Because of changing demand for paper due to shifts in reading habits and the increased use of tablet computers, obsolete or excess paper mills, which are fitted to supply large amounts of electricity and cooling water, can make good candidates for alternative sites for data centers.[2]

Other data centers have been built inside of underground caves[3] or on mountain tops,[4] and some have even been proposed to float in off-shore barges.[5] In each case, although the physical infrastructure of these facilities is quite different, the need to supply large amounts of electricity and ample capacity to remove the energy as waste heat are always common factors. It is the engineering of the power and cooling infrastructure that really distinguishes the efficiency of the data center.

Power Infrastructure

Although data centers may differ in mission, from providing network edge services to core compute to providing highly secure data processing and storage, in almost all cases, they require highly conditioned uninterruptable power to meet the high availability requirements their customers demand. The power to the data center needs to be highly conditioned to protect the servers, storage systems, and networking equipment in the data center from power transients. Both low-power conditions and power surges can cause equipment reliability issues and extended equipment downtime depending on duration and severity. Figure 9-1 shows a highly schematic layout of the connection of the electrical grid to the data center. When electrical grid power is interrupted, the uninterruptable power supply (UPS) assumes the load of the data center until the back-up generators can be started and reach full capacity. Due to cost constraints, it's typical to support only the computing equipment on an uninterruptable basis.

[1]See "Hamina, Finland," www.google.com/about/datacenters/inside/locations/hamina/.
[2]See Bart King, "Tablets are Significantly Reducing Media Industry's Paper Use" (2011), www.sustainablebrands.com/news_and_views/articles/tablets-are-significantly-reducing-media-industry%E2%80%99s-paper-use.
[3]See Nick Booth, "Green Mountain Puts Data Centre In NATO's Norwegian Cave" (2013), www.techweekeurope.co.uk/workspace/green-mountain-data-centre-norway-120063.
[4]See Rich Miller, "The World's Highest Data Center" (2013), www.datacenterknowledge.com/archives/2013/04/05/the-worlds-highest-data-center/.
[5]See Rich Miller, "Google Planning Offshore Data Barges" (2008), www.datacenterknowledge.com/archives/2008/09/06/google-planning-offshore-data-barges/.

www.dbooks.org

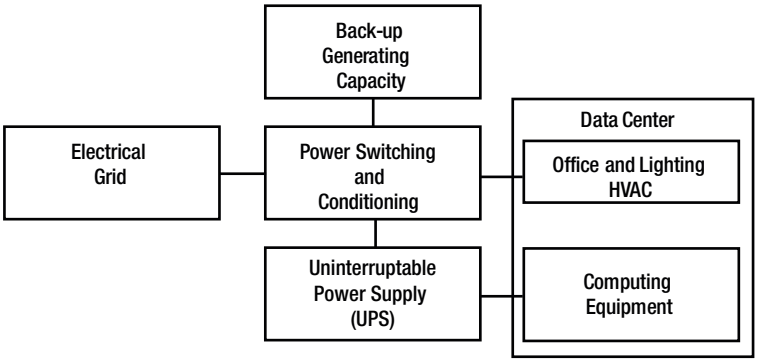


Figure 9-1. *Schematic of the connection of the electrical grid to the data center*

Power Distribution Efficiency

Although most of this book has concerned itself with the energy efficiency of the servers themselves, a key factor in overall data center efficiency is the energy loss in getting energy to the servers themselves. So-called *distribution losses* (bringing power from a remote generation facility to the site of use) can range from a few percent in cases where the load is close to the generating capacity—such as the large data centers in Quincy, Washington, and The Dalles, Oregon, which are within a few miles of hydroelectric energy sources—to 10%–20% when data centers are several hundred miles from electricity generation. Although these losses can be extremely important in overall efficiency, addressing them relies primarily on the siting of the data center facility, which is outside the scope of this book.

Power Conditioning

Power conditioning for the data center is among the most important missions of the facility. The power for the servers needs to be both "clean," meaning free from spikes or interruptions that might affect the availability of the servers in the data center, and economical, so the data center can achieve its mission at the lowest feasible cost. In this section, we'll look at the high-level topologies of two ways this can be achieved, in what are called AC and DC power distribution.

In *AC power distribution*, power to the server is supplied as AC voltage, typically at 208 VAC in the United States. AC power distribution is by far the most dominant in the industry. The flow of energy from the grid first powers the UPS system and then goes to the rack and row-level power distribution units (PDUs), where power is metered to the individual servers while at the same time protecting adjacent servers from electrical faults at any individual server.

Figure 9-2 shows a typical power distribution diagram for an AC data center. Power from the electrical grid feeds the UPS), which in turn provides clean uninterrupted power to the rack or row-level PDUs. These units convert power to levels usable by the servers while at the same time protecting the facility from faults at individual servers. The power and voltage conversions are highlighted. The UPS is shown with a bypass to allow more efficient operation.

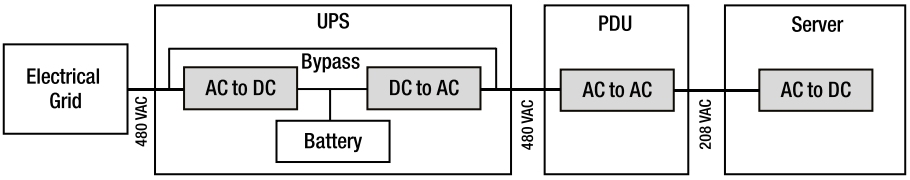


Figure 9-2. *A typical power distribution diagram for an AC data center*

One significant concern with this standard topology is the repeated conversion between AC and DC voltages between the grid and the server. Each conversion can results in a several percentage point sacrifice in efficiency, which can contribute to higher electricity costs.

Two of the conversions in the UPS—AC to DC and then back from DC to AC—can be eliminated by using a bypass or, more colloquially, *eco-mode* of the UPS. Although there may be concerns about the switch over time between the bypass and battery power in the event of a power failure, these concerns have been largely mitigated by technical development of suppliers. Typical switch-over times are now about one quarter of a power cycle, far below the damage threshold for the IT equipment in the data center. The Green Grid has adopted the use of a bypass as a "best known method."[6]

As an alternative to AC, there is compelling evidence that DC power to the data center can improve overall efficiency.[7] The primary efficiency gains come from the elimination of inefficient conversions from AC to DC. An example topology for this is shown in Figure 9-3. Power from the electrical grid feeds a converted PDU, which provides 48 VDC to the batteries and the servers. Voltage and AC to DC conversion steps are highlighted.

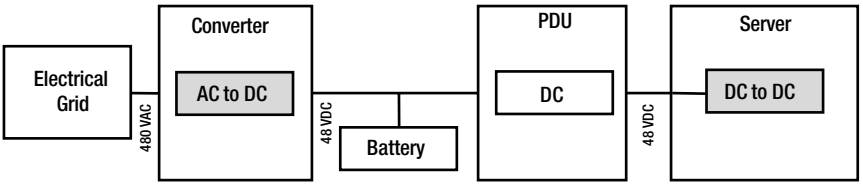


Figure 9-3. *A typical power distribution diagram for a DC data center*

The DC power infrastructure is inherently simpler than the AC power infrastructure because the number of required conversions, and hence expensive high-reliability high-power electrical gear, is reduced. However, the supply and experience base for AC power equipment is much larger since AC has been and remains the dominant industry

[6]See "Evaluation of Eco Mode in Uninterruptible Power Supply System" (2012), www.thegreengrid.org/en/Global/Content/white-papers/WP48-EvaluationofEcoModeinUninterruptiblePowerSupplySystems.

[7]See My Ton, Brian Fortenbery, and William Schudi, "DC Power for Improved Data Center Efficiency" (2008), http://hightech.lbl.gov/documents/data_centers/DCDemoFinalReport.pdf.

www.dbooks.org

standard. For instance, although DC can and has been used safely for years in the communications world, technicians in more traditional data centers are not currently trained to use it. Thus, although there may be some theoretical advantages of one approach over the other, the reality is that both high efficiency and low total cost can be achieved in both approaches provided the proper engineering practices, such as the use of UPS bypass in the AC case, are implemented.

Back-up Systems

Data center services need to be maintained even in the event of a power outage. For this, data centers rely on back-up power systems comprised, generally, of a short-term and longer term backup system. The short-term system is put into place to react quickly to fluctuations in supply and provide power, often for only a few minutes, until the high capacity longer term backup system can take over the full electrical load of the data center.

Uninterruptable power supplies (UPS) have long been built using large arrays of lead-acid batteries, and many data centers still use this simple but reliable technology. Lead-acid batteries, which are essentially like the battery in as typical automobile, can be purchased on the open market. The technology is mature and has not changed significantly in years. A downside to using batteries is that they need to be maintained and replaced on a regular basis to provide the intended high reliability of a back-up system.[8]

As a result, data center operators have sought and implemented alternative UPS schemes. One popular alternative is a high-speed rotating fly wheel system. In this case, rather than storing energy chemically, as in a battery, energy is stored kinetically through a high-speed, high-mass rotating flywheel. Flywheel systems take up less space than a battery system and require significantly less maintenance. But they also provide only a few seconds (in the range of 10–20, though this can vary depending on the size of the installation and load) of autonomy, implying that the back-up generators need to start up properly the first time. In addition, the size of the back-up generators needs to be increased slightly because, in addition to running the data center, they must also re-energize the flywheel system in a short amount of time to restore facility back-up.[9] Flywheel systems have the additional advantage that they run on AC supply, eliminating the needs for DC conversion.

The most common type of long-term back-up power for a data center is a diesel generator. The use of diesel generators is very common not only in data centers but also in other critical facilities such as hospitals, and thus they have a well-understood maintenance record and it is easy to find repair experts. Typically, diesel generators can represent about 10% of the capital cost of a data center. Although some proposals for reducing the cost of the generator capacity through intelligent IT have been made, these have not been widely adopted.[10]

---

[8]See "Implementing the Best Battery Maintenance Practices to Avoid Data Center Downtime" (2014), www.datacenterknowledge.com/archives/2014/05/05/implementing-best-battery-maintenance-practices-avoid-data-center-downtime/.

[9]See "Flywheel versus Battery in the Data Center" (2012), www.datacenterdynamics.com/focus/archive/2012/11/flywheel-versus-battery-data-center.

[10]Ibid.

There has been some innovation in backup power systems in data centers to avoid both cost and some of the downsides of having to operate large diesel generating plants (with the concomitant exhaust and noise) on a regular basis. For instance, the Facebook data center in Sweden forgoes about 70% of the typical back-up generator capacity because they are able to take advantage of redundant electrical grids in the area. This approach is brilliant and easily provides very high reliability to the power network at a very economical cost scale. However, redundant electrical grids are not common and thus this approach is of limited use in most cases.

Another alternative to diesel generators are fuel cells. Fuel cells convert fuel (typically either hydrogen or methane) to energy in an electrochemical reaction. Fuel cells have many advantages over other power sources. In addition to be relatively compact and clean, they can be brought close to the load, eliminating grid and distribution losses. Indeed, some data center operators have considered eliminating grid energy entirely for this reason, reporting favorable overall total cost of ownership under assumptions for realistic cost parameters.[11]

The eBay data center in Utah, commissioned in 2013,[12] gets most of its power from natural gas fuel cells built by Bloom Energy. According to eBay, the fuel cells reduce CO2 emissions about 50% and also increase the reliability of the data center. Indeed, the fuel cells are the primary source of energy for the data center, reducing costs associated with back-up generators and UPS systems.

Cooling Infrastructure

Providing power to the servers is an important side of the data center energy equation. On the opposite side, equally important, is the removal of all the waste heat generated by the servers. It's important to note that all the energy used to run the servers (and all the other equipment in the data center) needs to be removed as waste heat. That implies, for instance, that for a 10 megawatt data center load, exactly 10 megawatts of waste heat need to be dissipated to balance the energy input to the facility.

It's instructive to understand the evolution of data center cooling since it tells a strong story about advancing the infrastructure side of overall data center efficiency. Typical data centers build in the 1990s mainly relied on central computer room air conditioning (CRAC) units to provide cooling. Warm air coming from racks from servers was pulled into the CRAC units, chilled to temperatures as low as 60 degrees Fahrenheit, and pushed back out into the room through perforated raised floor tiles. This served adequately, though hot spots did turn up (often due to the poor uniformity of air circulation), which required special treatment. In addition, cold air entering the computer room was instantly mixed with warmer air, reducing the effectiveness of the cooling units.

[11]See "No More Electrical Infrastructure" (2013), http://research.microsoft.com/pubs/203898/FCDC.pdf.
[12]See "Introducing Our Salt Lake City Data Center" (2013), http://blog.ebay.com/introducing-our-salt-lake-city-data-center-advancing-our-commitment-to-cleaner-greener-commerce/.

www.dbooks.org

Figure 9-4 shows the layout of two data center types. The top figure is a typical "ballroom" configuration typical prior to the year 2000. Newer data centers, such as the one shown on the bottom of Figure 9-4, segregate warm and cold air and use local ambient air to economize operations.

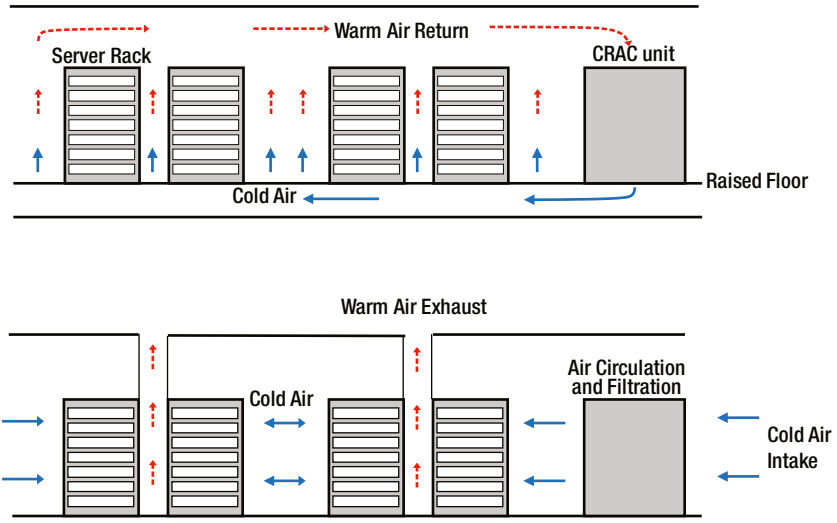


Figure 9-4. *Two data center types: A typical, old "ballroom" configuration (top), and today's data center (bottom)*

Starting in the early 2000s, engineers realized that segregating warm and cold air from each other by pushing cold air up and behind the server and then pulling the warm air out through the front of the server could improve the efficiency of the cooling units significantly. For instance, a study by T-Systems and Intel found that segregating hot and cold air could reduce the power usage effectiveness (PUE) of a model data center from a rating of 1.8 to around 1.3, a significant reduction in infrastructure energy use.[13] The work showed that even small air leaks, if not controlled, could have a significant effect on reducing the efficiency of the infrastructure.

The third major step in the evolution of data center infrastructure was the advent of what is called *free-air cooling* or the use of outside air to cool the data center. In many climate locations, outside air temperatures and humidity levels are low enough to effectively cool severs without additional help from air conditioners, provided air flows are maintained. There are many examples of data centers built to these standards, including the famous "chicken coop" data center, which relies on the tendency of warm air to rise where cross winds are, and then these move the warm air out of the building.[14] The industry

[13]See www.t-systems.com/news-media/white-papers/827826_2/blobBinary/White-Paper_Data-Center-2020-I.pdf.
[14]See David Filo, "Serving Up Greener Data Centers" (2009), https://yodel.yahoo.com/blogs/product-news/serving-greener-data-centers-1853.html.

has studied the capacity of different climates to support natural air cooling and studies have been published by the Green Grid where the effects of an updated AHRAE standard to allow wider temperature and humidity ranges were considered.[15]

The final stage of data center facility evolution is the advent of the high-temperature data center. Silicon and computer hardware components can tolerate higher temperatures than humans can. Studies have shown that even off-the-shelf components can operate safely at 40°C.[16] To further reduce cooling, there is a push to provide operational capability at 50°C.

Simplified Total Cost Models of Cost and Compute Infrastructure

A significant amount of work is available on total cost of ownership (TCO) models, including an excellent (and publicly available) one developed by Jonathan Koomey that gives significant detail for determining accurate cost benchmarks.[17]

In this section, rather than provide detailed models, we focus on a higher level perspective to help you understand the larger trends in cost. There is an inherent danger in using simplified models to make what can be relatively complex business decisions, but using them to help shape insight and recognize macroscopic trends can be useful.

Figure 9-5 shows a simplified TCO model of a 10,000 square foot data center. The model calculates the overall TCO of the entire data center, including building, infrastructure, and IT equipment costs. As important as what is in the model is what is not included. Important cost parameters like architectural choices, software licenses, labor and warranty coverage, insurance, and taxes could tip the conclusions of the model substantially and would need to be added for any responsible business decision.

[15]See "Updated Air-Side Free Cooling Maps" (2012), www.thegreengrid.org/en/Global/Content/white-papers/WP46-UpdatedAirsideFreeCoolingMaps-TheImpactofASHRAE2011AllowableRanges.

[16]See "The Efficient Datacenter" (2011), www.intel.com/content/dam/doc/technology-brief/efficient-datacenter-high-ambient-temperature-operation-brief.pdf.

[17]See Jonathan Koomey, "A Simple Model for Determining TCO for Data Centers" (2007), http://wdminc.com/whitepapers/SimpleModelDetermingTrueTCO.pdf.

www.dbooks.org

	Unit	Estimated Cost	Annualized Cost
Data Center			
Electrical and Cooling	Cost/Watt	$ 15	$4.0 M
10,000 Square Foot Building	Cost/Square Foot	$ 300	$0.20 M
Facility Depreciation	Years	15	--
PUE	(Unitless)	1.5	--
Server Cost	per Server	$ 5000	$5.0 M
Server Refresh Lifecycle	Years	5	--
Electricity Cost	per Kwh	$ 0.10	--
Server Average Power	Watts	300	$3.1 M
TOTAL	--	--	$12.3M

Figure 9-5. *A simplified TOC model of a 10,000 square foot data center calculated for nominal value*

At the most basic level, a cost model takes into account the capital costs (generally assumed to be incurred once and then amortized over some standard depreciation period to annualize the expense) and ongoing operating expense, which are paid on an as-used basis. Important parameters are highlighted in Figure 9-5. Figure 9-6 shows that operational electrical costs are a sizeable fraction of the overall TCO of a data center. Both the annual electrical energy cost and the electrical and cooling infrastructure scale directly with the watts consumed by the servers, thus a net reduction in server power can save on both operational and capital costs in the data center.

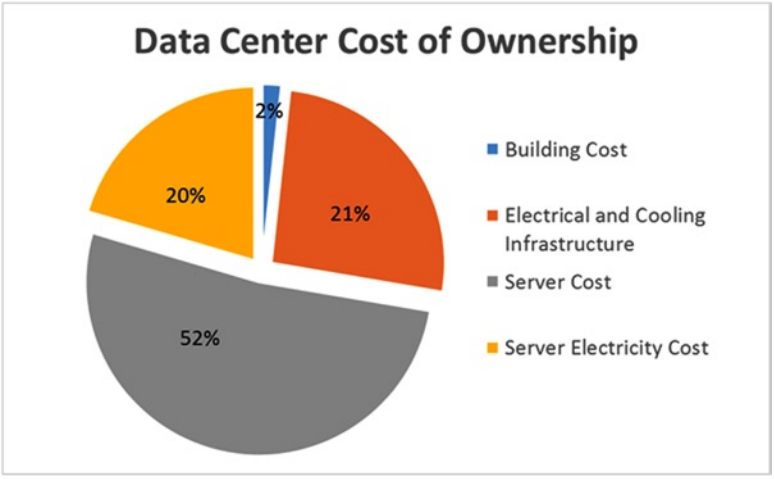


Figure 9-6. *The operational electrical costs are a sizeable fraction of the overall TCO of a data center*

The ranges of electrical infrastructure cost depend primarily on architectural choices as mentioned earlier. Decisions, such as the kind and number of back-up generators, electrical redundancy, layout, and location, can all affect the choices of costs widely. In general, the range of $8.00 to $20.00 per watt represents reasonable estimated bounds, though deviations both above and below this range are possible.

Data center efficiencies in the range from PUE = 1.1 up to 3.0, where PUE is the power usage effectiveness, as defined in Chapter 1, are known in the industry. Generally the PUE in older facilities is much higher than it is in more modern facilities that use ambient cooling to reduce overhead costs.

Performance per Watt per Dollar

It's common to hear people concerned about data center costs talk in terms of "performance per watt per dollar," yet there seems to be no good description of this phrase in published literature. In this section, we briefly discuss where terms with the units of performance per watt per dollar come in to the cost of ownership equations.

In order for you to understand this, we need to add one more dimension, which we call *computational work*. Computational work is not the same a physical work, but it can nevertheless be thought of in a similar way—making a physical change on a system (in the case of a computer, the bits). This physical change requires energy, and thus the energy required to make the change can be equated to work done.

For the sake of the present case, we'll consider the computational work rate to be done by a data center as a number, T, of transactions per second. The specific type of transaction isn't important, and we'll make a simplifying assumption that these transactions are uniform. If the capacity of the server, called its performance, is p transactions per second, then the number of servers, N, required in the data center is just

$$N = T/p$$

Now the total power use of the data center, P_{total}, will be the power required by the server, P_{server}, plus the power used by the infrastructure, or

$$P_{total} = N * P_{server} * PUE$$

Which, upon substituting the relationship above, becomes

$$P_{total} = T * P_{server} * PUE/p$$

Noting that Energy equals Power * Time, we can annualize the costs by considering the Power and the transactions rate, T, over a standard interval of time, which we'll take to be one year. When you perform some simple algebra, it is easy to calculate cost efficiency in terms of transaction per dollar of operational energy cost:

$$T/(P_{total} * \text{Energy Cost}) = p/(P_{server} * \text{Energy Cost} * PUE)$$

www.dbooks.org

Note that the period of time over which the costs are averaged cancels in both the numerator and denominator, as we'd expect. The right-hand side has units of performance per watt per dollar. This equation has intuitive appeal. To maximize operational efficiency, you would want to maximize the transactions per energy cost. The equation highlights that this is achieved by maximizing system performance; minimizing server power and energy costs; and maximizing the data center infrastructure efficiency. Maximizing performance per watt per dollar minimizes the cost per transaction.

Summary

In this chapter we have shown that maximizing data center efficiency resolves to maximizing both the energy efficiency of the servers and the data center infrastructure. Power distribution plays a large role in the management of overall data center efficiency. Although AC dominates the current design and distribution of power to data centers, DC offers equivalent efficiency with fewer power conversions. As data centers move to alternative power sources like solar and fuel cells, which inherently provide DC power, we can expect to see a greater foothold of DC in data center design.

Finally, we have shown that total cost of ownership can offer a complex set of trade-offs in optimizing overall data center design, with both server performance and efficiency gains offering some of the most powerful variables in the overall optimization. This optimization reduces to maximizing "performance per watt per dollar" in order to achieve maximum cost efficiency.

APPENDIX A

Technology and Terms

AC	Alternating current.
ACNT	A term commonly used to describe the IA32_APERF MSR used to calculate average frequency over a user-defined time window.
ACPI	The Advanced Configuration and Power Interface is used by BIOS to expose platform power management capabilities.
APIC	Advanced Programmable Interrupt Controller.
AR	The application ratio is used to describe the CPU logic switching rate of a workload.
ASHRAE	American Society of Heating, Refrigerating, and Air-Conditioning Engineers.
ASPM	Active State Power Management is a feature used to manage the power of PCIe links.
Avoton	Codename for Atom C2000-series SoC that follows Centerton.
AVX	Advanced vector extensions are integer and floating-point instructions used to improve performance.
Bin	A term used to describe the increase in frequency between any two P-states P*n* and P*n*-1.
BIOS	Basic Input/Output System refers to the firmware used to initialize a server.
BMC	The baseboard management controller is a dedicated microcontroller that provides remote monitoring and management functionality.
CC0/CC1/CC3/CC6	Describes a specific core-level C-state.
Centerton	Codename for Atom S1200-series processor.

(*continued*)

www.dbooks.org

CKE	The clock enable signal is commonly used to identify rank-level power down modes for memory.
CLST	Closed Loop System Throttling is a power management feature that enables hardware protection using Node Manager and a PMBus power supply.
CLTT	Closed Loop Thermal Throttling utilizes temperature monitoring to manage memory thermal throttling.
CPI	Cycles per instruction is a basic performance metric.
CPU	Central processing unit.
CPUID	CPU identification instruction used to discover processor type and features.
CRAC	Computer room air conditioner.
CRAH	Computer room air handler.
CSR	A control and status register frequently used for power management monitoring or control.
C-state	An idle state where the processor has halted execution of instructions.
DC	Direct current.
DPC	The DIMMs per channel is a ratio used to describe the memory population of a platform.
DRAM	Dynamic Random Access Memory.
D-state	A low-power idle state for devices (PCIe, SATA).
DTSMAX	The maximum allowed temperature of the processor.
Dynamic Switching	A processor power management feature that automatically switches a platform to performance mode when capacity is high.
EBS	Event-based sampling is a monitoring technique that allows operators to associate power and performance events with the specific modules, functions, and lines of code that caused them.
ECC	Memory error correction used to provide protection from both transient errors and device failures.
EDP	The electrical design point
EEE	Energy efficient Ethernet is a low power mode that reduces PHY power.

EN	Identifies a platform or processor as Entry Level (Xeon E3 for one-socket servers).
Energy Perf Bias	A model-specific register used to control how aggressively power management features will be used.
Entry Latency	The time it takes to transition from an active to idle state (typically measured in microseconds).
EP	Identifies a platform or processor as efficient performance (Xeon E5 for two- to four-socket servers).
EPA	Environmental Protection Agency, responsible for the Energy Star program.
EX	Identifies a platform or processors as expandable (Xeon E7 for 4-socket and larger servers).
Exit Latency	The time it takes to transition from an idle to active state (typically measured in microseconds).
FIVR	A Fully Integrated Voltage Regulator is a high-current switching regulator integrated into the processor.
G-state	A global state that identifies the overall power state of a platform.
Haswell	Codename for the Xeon E5 v3 processor that follows Ivy Bridge.
HDD	A hard disk drive is a traditional spinning hard drive.
HIS	Integrated heat spreader.
HLT	Halt instruction used by an operating system to enter a C1 state.
HPC	High performance computing.
HSC	Hot swap controller.
HT	Hyper-Threading technology is Intel's implementation of simultaneous multithreading.
I/O	Input/output is used to describe capabilities for communication such as DDR, PCIe, and coherent interconnects such as QPI.
IA	The Intel Architecture term is commonly used to identify a hardware feature unique to Intel products.
IB	InfiniBand is a low-latency and high-throughput communications link frequently used in high performance computing.

(*continued*)

www.dbooks.org

ICCMAX	The maximum current power delivery a platform can supply.
IMON	A voltage regulator current monitor used to measure power.
IPMI	The Intelligent Platform Management Interface is a specification and operating system independent interface for remote management.
ITD	Inverse temperature dependence.
Ivy Bridge	Codename for the Xeon E5 v2 processor that follows Sandy Bridge.
LDO	Low-dropout regulators used to provide variable voltages across cores in a processor with a single input voltage.
Linpack	An HPC benchmark derived from a collection of Fortran linear algebra routines.
L-state	A low-power idle state for interconnects (PCIe, DMI, QPI).
MBVR	A motherboard voltage regulator.
MCNT	A term commonly used to describe the IA32_MPERF MSR used to calculate average frequency over a user-defined time window.
MCP	A multi-chip package is where multiple chips are integrated together in the same package.
ME	The (Intel) Management Engine in the Platform Controller Hub used for monitoring, power capping, and hardware protection.
MMIO	Memory mapped I/O.
MSR	A model-specific register frequently used for power management monitoring or control.
MWAIT	A Monitor Wait instruction used by an operating system to enter a C1 or deeper C-state.
Nehalem	Codename for the Xeon 5500 processor.
NTB	Non-transparent bridging is a support technology used to create non-coherent interconnects between nodes using PCIe.
NUMA	Non-uniform memory access allocation provides contiguous memory regions for each processor's local memory.

NVMe	Non-Volatile Memory Express is a specification for directly connecting SSDs on PCIe that provides lower latency and higher performance than SAS and SATA.
OEM	Original equipment manufacturer.
OLTP	Online transaction processing.
OLTT	Open loop thermal throttling utilizes a static bandwidth limit to manage memory thermal throttling.
OS	Operating system.
OSPM	Operating system power management is a term commonly used to describe operating system power management policies and device drivers.
P1 frequency	Represents the CPU base frequency, guaranteed frequency, or the marked frequency of a CPU.
Path Length	Path length is a basic performance metric that measures the average number of instructions it takes to complete a single unit of work.
PC0/PC1/PC2/ PC3/PC6	Describes a specific package-level C-state.
PCH	Also known as South Bridge, Platform Controller Hub is a chipset connected to the processor that integrates many features that would otherwise require discrete controllers (such as storage, network, USB, management, and legacy).
PCIe	Peripheral Component Interconnect Express is a high-speed serial communication bus.
PCM	Performance Counter Monitor is a set of stand-alone tools used to collect core and uncore power and performance events.
PCPS	Per-core P-states allows individual cores that can each operate at their own frequency and voltage independent of what P-state other cores are in.
PCU	The power control unit is an internal microcontroller used to facilitate CPU power management.
PECI	The Platform Environment Control Interface is an interface for management controllers to communicate with the CPU.
PL1/PL2/PL3	A power level indicates a specific power limit used for power capping and power delivery protection.

(*continued*)

www.dbooks.org

P-Limit (I/O)	A power management feature that allows the uncore to autonomously increase its uncore P-state to improve PCIe performance.
P-Limit (perf)	A power management feature that allows an idle socket to increase its uncore P-state to improve snoop and memory latency.
PLL	Phase-locked loop used to drive clocks.
PMBus	Power Management Bus is an open standard protocol used for power management of power supplies.
PMIC	A power management integrated circuit is applied to integrated circuits that have multiple power conversion controllers in one small package.
P*n* frequency	Represents the lowest frequency P-state or the most energy efficient frequency.
P-state	A performance state is an active state that represents a fixed frequency and voltage operating point.
PSU	Power supply unit.
PUE	Power usage effectiveness is defined as the ratio of the total energy use by the datacenter to that of the energy used by the IT equipment.
PWM	Pulse-width modulation.
QPI	QuickPath Interconnect is used for multi-socket communication.
RAPL	Running Average Power Limit is a power management feature used to maximize performance while meeting a specific thermal or power constraint.
RDTSC	Read time stamp counter instruction used by software to measure time.
Sandy Bridge	Codename for the Xeon E5 processor that follows Westmere.
SAS	Serial attached SCSI is a common protocol for connecting disks to a storage controller.
SATA	Serial ATA is a common protocol for connecting disks to a storage controller.
SEL	The System Event Log is a centralized event log used by management firmware.

Self-refresh	A low power memory power state where the DIMM itself is responsible for handling refresh.
SKU	The stock keeping unit term is commonly used to identify a CPU by its specific features (microarchitecture, core count, frequency, TDP).
SmaRT	Smart Ride Through is a technology that allows a server to function through momentary loss of AC power.
SMBus	The System Management Bus enables lightweight communication between platform devices.
SMT	Simultaneous multithreading.
SoC	A system on a chip is the coupling of the CPU with special-function hardware components in the same die.
SPEC	The Standard Performance Evaluation Corporation creates and maintains server benchmarks.
SRAM	Static random access memory.
SSD	Solid-state disk drive is a high-performance hard drive that stores data in flash memory chips.
S-state	A sleep state that powers down most platform components.
SVID	Serial VID is a serial communication bus between the processor package and the voltage regulator controllers.
TC0/TC1/TC3/TC6	Describes a specific thread-level C-state.
TCO	Total cost of ownership is a metric that estimates both the direct and indirect costs of a system.
TDP	A thermal design point specifies the amount of power that the CPU can consume, and therefore the amount of heat that the platform must be able to remove in order to avoid thermal throttling conditions.
THERMTRIP	A term used to describe the catastrophic trip temperature that, when exceeded, will result in immediate hardware shutdown.
TIM	Thermal interface material fills the air gaps between the component being cooled and a heat sink.
Tj	The junction temperature describes the internal temperature of the die.
TPC	The Transaction Processing Performance Council creates and maintains server benchmarks.

(*continued*)

www.dbooks.org

TSC	Time Stamp Counter.
TSOD	Thermal sensor on die is a thermal sensor used in memory to measure temperature.
T-state	An active state where core execution is duty-cycled at a fixed interval for thermal, electrical, or power reasons.
Turbo frequency	Represents opportunistic frequency range about the CPU base frequency.
UFS	Uncore Frequency Scaling describes a power management feature that allows the uncore to maintain its own P-state.
UMA	Uniform memory access allocation interleaves every other cache line across each processor's local memory.
Uncore	A term commonly used to describe processor on-die logic outside of the cores.
USB	Universal Serial Bus.
Vmin	The minimum voltage used for the lowest frequency P-state.
VMM	Virtual machine monitor.
VR	Voltage regulator.
Vret	The retention voltage required to maintain state in a circuit.
VT	Virtualization technology is a term commonly used to describe technologies used to improve performance in a virtualized environment.
Westmere	Codename for the Xeon 5600 processor that follows Nehalem.
Xeon E3	Processor type used in one-socket servers for workloads with low compute requirements.
Xeon E5	Processor type used in two-socket servers for most workloads.
Xeon E7	Processor type used in 2-socket to 256-socket servers for mission critical and scale-up workloads.
Xeon Phi	Coprocessors used to accelerate workstation and cluster performance typically used in HPC.

Index

A

AC/DC power supply
  boost stage, 123
  converter, 122
  isolated buck stage, 123
  MOSFETs, 123
  PDUs, 309
  PMBus, 124
  redundance, 124
  shared power, 124
ACPI. *See* Advanced configuration and power interface (ACPI)
ACPI interfaces
  C-states (CST), 162
  FFH, 161
  get hardware limit (GHL), 171
  original equipment manufacturer (OEM), 161
  OSC and PDC, 159
  performance control (PCT), 161
  performance supported states (PSS), 160
  power averaging interval (PAI), 171
  power impact, 161
  power meter capabilities (PMC), 171
  power meter measurement (PMM), 171
  power trip points (PTP), 171
  P-state domain (PSD), 160
  set hardware limit (SHL), 171
Active power down (APD), 79
Active State Power Management (ASPM), 110
Advanced configuration and power interface (ACPI), 154, 156
Aggressive Link Power Management (ALPM), 302
APD. *See* Active power down (APD)
Architecture, Turbo
  consumer devices, 54
  C-states, 55–56
  electrical protection, 55
  fused frequencies, 56
  power/thermal limits, 54
  thermal protection, 54
ASHRAE, 5
ASPM. *See* Active State Power Management (ASPM)

B

Baseboard management controller (BMC), 155, 230–231
Basic input/output system (BIOS), 285
BIOS firmware
  ACPI power states and transitions, 157
  BMC, 155
  CPUID, 155
  CSRs, 154
  C-states, 158
  definition, 154
  D-states, 159
  IPMI, 154
  microcode update, 155
  MMIO, 154
  MSRs, 154
  operating system (OS), 154
  PCU, 154
  PMCSR, 159
  P-states, 158
  RDMSR and WRMSR, 154
  setup utility, 162
  S-states, 157
  TLBs, 156
BL8. *See* Burst-Length 8 (BL8)
BMC. *See* Baseboard management controller (BMC)
Burst-Length 8 (BL8), 72

www.dbooks.org

C

Central processing unit (CPU)
  optimization, 305
  power management, 285
  thermal management, 302
Chipset integration, 96
Clock-enabled (CKE) power savings, 298
Closed loop system protection (CLST), 163
Closed loop system throttling (CLST), 135
Closed-loop thermal throttling (CLTT), 84
CLTT. *See* Closed-loop thermal throttling (CLTT)
Computer room air conditioning (CRAC) units, 312
Control and status registers (CSRs), 154
Core offline, 292
Core Parking, 293
CPU architecture
  8c Atom Avoton and 10c Xeon Ivy Bridge EP, 26
  Avoton SoC and Sandy Bridge packages, 27
  cache hierarchies, 25
  components, 22
  digital synchronous logic and clocks, 31
  external communication, 29
  Intel server processors, 33
  I/O circuits, 32
  on-die fabrics and uncore, 27–28
  power control unit (PCU), 28–29
  SRAM and eDRAM, 32
  thermal design, 30
  threads, cores and modules, 23
  total cost of ownership (TCO), 21
CPU integration, 95–96
CPU power management
  breakdown
    I/O devices, 35
    logic power, 35
  common terms, 34
  C-states
    core C-states, 44–45, 47
    module, 49
    package, 47
    thread, 44
  description, 33, 41
  frequency, voltage and temperature, 36–37
  P-states
    base clock frequency (bclk), 49
    per core P-states (PCPS), 51–52
    per socket P-states (PSPS), 51
    server generations, 50
    uncore frequency scaling, 53
    voltage regulators (VRs), 50
    web server, 49
  S0ix, 58
  S-states and G-states, 57
  T-states, 56
  Turbo (*see* Architecture, Turbo)
CPU sockets
  DP platforms, 97
  MP nodes, 97
  node controllers, 98–99
  UP server systems, 98
CPU thermal management, 66, 68–69
CSRs. *See* Control and status registers (CSRs)
C-states
  BIOS, 291
  core parking, 293
  DDR memory, 293
  Linux, 291
  logical processors, 292
  operating system, 293
  power savings, 291
  system utilizations, 291
  turbo frequencies, 292
  workloads, idle, 294
Customer relationship management (CRM) systems, 275

D

Data centers
  AC power distribution, 309
  air flow management systems, 5
  ASHRAE, 5
  back-up power systems, 311–312
  “bottoms-up” methodology, 2
  chicken coop, 313
  computational work, 316
  cooling systems, 5, 312–314
  DC power distribution, 310
  diesel generators, 311
  digital economy, 1
  eBay, 1

electrical grid connection, 308–309
energy consumption, 19
energy efficiency and cost, 16, 307–308
energy proportional computing, 5–6
Facebook, 1
fly wheel system, 311
free-air cooling, 5, 313
fuel cells, 312
Green Grid, 4
Internet, 2
IT equipment, 3
lead-acid batteries, 311
Moore's law, 19
natural disasters, sheltering, 307
network edge services, 308
power and cooling infrastructure, 18, 308
power back-up systems, 2
power conditioning, 309–311
power consumption, 3
power transients, 308
PUE, 4, 18, 313, 316
scale and architecture, 307
server, heat removal, 312
servers, 2
TCO models, 16, 18
total power, 316
underground caves/mountain tops, 308
UPS, 308, 311
Data manipulation, 173
DC to DC power converters
motherboard linear regulators, 127
motherboard multiphase buck converters, 125–126
PMIC, 128
single-phase buck converters, 125
SVID, 126
voltage regulators, 127–128
DDR thermal management
CLTT, 84
MEMHOT, 84
memory throttling, 83
OLTT, 83
SMBus, 83
monitoring, 83
TSOD, 83
Device drivers, 193–194
DIMMs, 71–74, 77, 80–82, 84
Distributed power management (DPM), 206
Distribution losses, 309
DMA coalescing, 301

E

ECC. *See* Error correcting code (ECC)
EEE. *See* Energy Efficient Ethernet (EEE)
Energy efficiency programs, 275
Energy Efficient Ethernet (EEE), 109
Energy Performance Bias (EPB), 294–295
Enterprise resource planning (ERP) systems, 275
Environmental Protection Agency (EPA), 6–8, 275
Error correcting code (ECC), 72
Exascalar method, 14, 16
Execution consolidation
energy efficiency, 190
power capping, 191
single-threaded performance, 191

F

FFH. *See* Functional fixed hardware (FFH)
FIVR. *See* Fully Integrated Voltage Regulator (FIVR)
Frequency management algorithms, 286
Fully Integrated Voltage Regulator (FIVR), 127–128
Functional fixed hardware (FFH), 161

G

Green Grid, 4, 310, 314

H

Hard disk drives (HDDs), 114
Hardware-controlled performance (HWP), 180–181
Hardware monitoring
core performance monitors, 210–211
counter access and constraints, 214
CPI, 215
C-state statistics, 222–223
edge detection and average time, state, 212

www.dbooks.org

Hardware monitoring (*cont.*)
energy use, 215–216
events and metrics, 214
fixed counters, 210
frequency and voltage, 219–221
global freeze/unfreeze, 212
IPMItool, 262, 264–265
Linux perf, 259
Eclipse perf/sysprof, 258
integrated profiling and tracing subsystem, 258
logical processor activity, 260
malloc function, 261
perf top, 259
qemu-system process, 261
software thread activity, 260–261
Xeon E5/E7 processors, 262
memory power and performance statistics, 225
path length, 215
PCIe power management, 226
PCM. *See* Intel performance counter monitor (PCM)
QPI power management and performance, 226, 228–229
RDTSC, 215
standard and occupancy events, 213
status snapshots, 213
temperature, 217–218
types, 210
uncore performance monitoring, 212
HDDs. *See* Hard disk drives (HDDs)
High performance computing (HPC)
DNA decoding, 14
exascalar method, 14–16
Green500, 14
power consumption, 14
TCO, 15
HPC Linpack, 8
Hyperthreading (HT), 295

I

Industry workloads
ERP and CRM systems, 276
network based, 275
server, characterization and optimization, 275
SPECpower, 276
TPC, 276
Integrated graphics, 194
Intelligent platform management interface (IPMI), 154, 200
Intel Memory Latency Checker (Intel MLC) tools, 273
Intel performance counter monitor (PCM)
four-socket system, 255–256
frequency clipping cause, 257
frequency transitions, 257
Xeon E5/E7 processors, 256–257
Intel server processors, 33
Intel software developers manual (SDM), 210
Intel Xeon processors, 9
Interrupt moderation, 110
IPMI. *See* Intelligent platform management interface (IPMI)

J, K

Java Virtual Machine (JVM), 274

L

LAN power management
ASPM, 110
balance network traffic, 109
EEE, 109
interrupt moderation, 110
media speed, 109
WoL, 110
Linux distributions
CPUidle infrastructure, 204
IA32_ENERGY_PERF_BIAS, 204
Intel, 204
kernel version, 205
ondemand, 203–204
Linux kernels, 182, 205
Load line, 274

M

Management controllers monitoring
BMC and ME, 230
power sensors, 230
sensors, 231–235
synthetic sensors, 231
Management Engine (ME), 230
Management firmware
BMC, 163

CLST, 163
definition, 163
hardware protection, 163
IPMI, 168
node manager (*see* Node manager)
power capping, 164
sensor model (*see* Sensor model)
smart ride through (SmaRT), 163
system event log (SEL), 169
Memory
buffer chips, 99
capacities, 99
DRAM devices, 145
management, 192–193
optimization, 305
power management, 298
reliability features, 85
risers, 99
thermal sensors, 144
Memory-mapped input/output (MMIO), 154
Microservers, 96
Microsoft Windows Server
core parking, 202
C-state policy, 201
memory cooling, 202
power management features and improvements, 202
power saver, balanced and high performance, 201
P-state policy, 201
MMIO. *See* Memory-mapped input/output (MMIO)
Model specific registers (MSRs), 154
Monitoring
description, 209
management firmware
hot swap controllers (HSCs), 163
power supply units (PSUs), 163
voltage regulators (VRs), 163
sensor measurements, 209
system and subcomponent monitoring, 209
Moore's law, 19
Motherboard linear regulators, 127
Motherboard multiphase buck converters, 125–126
Motherboard voltage regulators (VRs)
burst mode, 132
diode emulation mode, 132
losses, 131
phase shedding, 131–132
power losses, 129
single-phase buck converter, 129–130
MSRs. *See* Model specific registers (MSRs)
Multi-chip package (MCP), 27

N

Networking
ambient temperature, 106
attached media, 108–109
description, 105
frequency/voltage, 112
LAN component power, 106
LAN power management features (*see* LAN power management)
link power states, 111–112
TDP, 107
thermal management, 106, 108
USB connectivity, 111
Network interface card (NIC), 300–301, 305
Node manager
API, 170
attributes of, 166
BMC, 163
capabilities, 163
external interfaces and components, 165
high performance computing (HPC) environments, 170
policies, 165
Non-uniform memory access (NUMA), 78, 189, 246
Non-volatile memory express (NVMe), 117–118
NUMA. *See* Non-uniform memory access (NUMA)
NVMe. *See* Non-volatile memory express (NVMe)

O

OLTT. *See* open-loop thermal throttling (OLTT)
On-die termination (ODT), 80
Open-loop thermal throttling (OLTT), 83
Operating system capabilities (OSC), 159
Operating system (idle), 276–277

www.dbooks.org

Operating system monitoring tools
Perfmon and Logman, 266–268
SAR, 265–266
Operating system power management (OSPM)
collaborative interface, 180–181
C-state control, 174
C-state policy, 176–177
firmware control, 182–183
IA32_ENERGY_PERF_BIAS MSR, 187
IA32_PERF_CTL MSR (0x199), 179
IA32_PERF_STATUS MSR, 179–180
MWAIT, 175
performance capacity (*see* Performance capacity)
performance impact, 174
phones and tablets, 174
power state coordination, 186
processor utilization, 178
process scheduling, 188
P-state policy, 183
timer tick frequency, 189
topology and capability awareness, 188
T-state control, 187
Windows Server control panel, 187
Opportunistic self-refresh (OSR), 299
OSPM. *See* Operating system power management (OSPM)

P

Patrol Scrub, 300
PCM. *See* Intel performance counter monitor (PCM)
Performance capacity
average utilization, 184
frequency, 184
processor utilization, 183
P-states, 186
Performance metrics, 283–285, 303
Performance per watt per dollar, 316
Peripheral Component Interconnect Express (PCIe), 296–297
Physical address (PA), 72
Platform Controller Hub (PCH)
architecture block diagram, 101
capabilities, 100
components, 101–102
PCIe, 105
phase-locked loop, 105
platform power management, 102, 104
and TDP power, 104
thermal management, 105
three-chip solution, 100
two-chip Xeon Intel Architecture, 100
usage configurations, 104
Platform power components, 93–95
PMCSR. *See* Power management control and status register (PMCSR)
PMIC. *See* Power Management Integrated Circuit (PMIC)
Power conversion losses
energy transmission, 128
fixed losses, 128
motherboard VRs, 129–132
proportional losses, 128
system power supplies, 133–137
types, 129
Power delivery
AC/DC power supply (*see* AC/DC power supply)
block diagram—loads, 121
components, 120
conversion losses (*see* Power conversion losses)
DC/DC power converters, 120, 124–128
description, 118
dual socket power conversion, 119
energy transmitting, 122
motherboard, 122
Power distribution. *See* Data centers
Power management
algorithms, 285
APD, 79
CKE generation, 81
CKE power savings, 79
definition, 71
device power characteristics, 75
D-states, 90
hot-add flows, 90
link frequency/voltage, 89
link power states, 86
link width, 90
ODT, 80
PCIe, 87
performance, 79
PPD, 79
RAPL, 84
self-refresh, 81
voltage/frequency, 82

Power management control and status register (PMCSR), 159
Power management integrated circuit (PMIC), 128
Power metrics, 281–283
Power-saving techniques
  strategies, 40
  turn it down, 39
  turn it off, 38
Power usage effectiveness (PUE), 4, 313
PPD. *See* Precharge power down (PPD)
Precharge power down (PPD), 79
Prefetchers, 296
Processor driver capabilities (PDC), 159
Process scheduling, 188
Prochot, 69
P-states. *See* Turbo
PUE. *See* Power usage effectiveness (PUE)

Q

Quick path interconnect (QPI), 297

R

RAPL. *See* Running average power limit (RAPL)
Reliability, availability and serviceability (RAS) features, 269
Running average power limit (RAPL), 84
  capabilities, product generations, 59–60
  components/constraints, 58–59
  DRAM, 65
  IMON and digital power meter, 62
  Linpack (HPL), 63–64
  power-throttling, Turbo 2.0, 60–61
  Sandy Bridge, 60

S

SAS. *See* Serial attached SCSI (SAS)
SATA. *See* Serial advanced technology attachment (SATA)
Self-refresh, 299
Sensor model
  field replaceable unit (FRU), 169
  sensor data records (SDRs), 169
Serial advanced technology attachment (SATA), 114–116
Serial attached SCSI (SAS), 114–116
Server chipsets
  description, 100
  legacy capabilities, 100
  PCH (*see* Platform Controller Hub (PCH))
  SoCs integrate, 100
Server CPU architecture. *See* CPU architecture
Server Efficiency Rating Tool (SERT), 273, 275
Servers
  hardware and software configuration, 269
  I/O devices, 269
  operating systems, 269
  optimizing steps, 270–271
  RAS features, 269
Server-side Jave operations (ssj_ops), 11–12
Server system on a chip (SoC), 96
Silicon process technology, 11, 13
Simultaneous multi-threading (SMT), 188
Single-phase buck converters, 125, 129
SMBus. *See* System Management Bus (SMBus)
Software components, power management, 153
Software computation, 173
Software monitoring
  applications, 235
  C-state events, 242
  frequent network interrupt handling, 236
  hardware C-state residency, 241
  interruption, 244–245
  I/O performance, 247, 249
  kernel time, 235
  logical processor level, 241
  memory, 245–246
  Powercfg (Windows), 253–254
  PowerTOP (Linux), 251
  products and versions, 236
  P-state events, 242–243
  scheduler, processes and threads, 243–244
  simultaneous multithreading (SMT), 238–239
  software C-state residency, 240

www.dbooks.org

Software monitoring (*cont.*)
Turbostat (Linux), 250–251
utilization and processor time, 236–238
Solid state drives (SSDs), 114
SPEC. *See* Standard Performance Evaluation Corporation (SPEC)
SPECPower
dual socket servers, 9
Intel's "tick-tock" model, 11
Java, 8
load line, 8, 10
power ratios, 8
server, 8
SRAM. *See* Static random-access memory
SSDs. *See* Solid state drives (SSDs)
Standard Performance Evaluation Corporation (SPEC), 8, 274
Static random-access memory (SRAM), 32
Storage
cold storage system, 113
compute nodes, 112
description, 113
frequency/voltage, 116–117
HDDs and SSDs, 114
NVMe power states, 117–118
power consumption, 113
SAS and SATA, 114–117
servers, 113
Storage optimization, 305
Storage power management, 301–302
System characterization
analysis, 281
collection frequency, 279
data collection, 278–279
event ordering and groups, 280
methodology, 280–281
steady state *vs.* non-steady state, 277–278
tools, 280
System Management Bus (SMBus), 83
System memory
architecture, 71
BL8, 72
capacity, 74
CPU DDRIO, 84
CPU interconnect (QPI), 85
CPU I/Os, 85
DDR channels, 73
DDR, CPU platform, 72
DDR3 and DDR4, 76
DDR4 DIMM, 75
DDR generation, 76
definition, 71
devices, 72
dual-ranked (DR), 73
ECC, 72, 74
imbalanced memory, 78
LRDIMMs, 77
memory thermal management techniques, 83
NUMA, 78
oct-rank (OR), 73
PA, 72
quad-ranked (QR), 73
RDIMMs, 77
SODIMMs, 77
UDIMMs, 77
UMA, 78
System power supplies
CLST, 135
cold redundancy, 137
different size, 134–135
losses, 133, 135–136
750 W PSU efficiency, 133–134
750 W PSU losses, 133
wattage ratings, 133

T

TCO. *See* Total cost of ownership (TCO)
TDP Xeon processors, 286
Technology and terms, 319–325
Thermal control inputs—sensors
layout, 148
platform component, 147
power supplies, 149
types, 146
voltage regulators, 148–149
Thermal gradient, 84
Thermal management
air-cooled system, 140
air heating, 139
component temperature specifications, 137
CPU packaging, 140
customer's requirements, 145
equation, 140
fan speed control, 149–150
heating types, 138

heat sink, 140–141
heat transfer terms, 139
IHS, 139
inputs—sensors, 146–149
local ambient, 139
memory, 144–145
natural temperature variation, 145
processors, 143–144
server cooling system, 137
system considerations, 141–143
TIM, 139
VRs, 149
worst-case corner, 138
Thermal sensor on-die (TSOD), 83, 217
Total cost of ownership (TCO)
models, 15, 314–316
Transaction Processing Performance
Council (TPC), 276
TSOD. *See* Thermal sensor on-die (TSOD)
Turbo
BIOS, 287
"burst" performance, 286
frequency control
Linux, 288
Windows, 287–288
modern operating systems, 286
ratio limits, 289–290
UFS, 290
voltage and frequency, 286

U

UMA. *See* Uniform memory access (UMA)
Uncore frequency scaling (UFS), 290
Uniform memory access (UMA), 78
Uninterruptable power
supply (UPS), 308–309, 311

V

Virtual machine monitor (VMM)
energy efficiency, 198
hypervisor model, 195
idle scenarios, 195–196
logical processor utilization, 197–198
migration, 199–200
power state control, 195
server consolidation, 198
software/hardware
enhancements, 198
VMM. *See* Virtual machine monitor
(VMM)
VMWare ESX/ESXi, 206–207

W, X, Y, Z

Wake on LAN (WoL), 110
WoL. *See* Wake on LAN (WoL)
Workloads
characterization and
optimization, 272
data collection, 271
energy efficiency, 273
Intel MLC, 273
JVM, 274
load line, 274
NetPIPE, 273
power and performance, 272–273
SERT, 275
software services, 271
SPECpower, 274
testing tools, 273
transactions/computations, 272
types, 273
virtual machines, 272

www.dbooks.org

Energy Efficient Servers

Blueprints for Data Center Optimization

Corey Gough
Ian Steiner
Winston Saunders

Apress open

Energy Efficient Servers: Blueprints for Data Center Optimization

Corey Gough, Ian Steiner, Winston Saunders

Copyright © 2015 by Apress Media, LLC, all rights reserved

ApressOpen Rights: You have the right to copy, use and distribute this Work in its entirety, electronically without modification, for non-commercial purposes only.

License for Distribution of the Work: This Work is copyrighted by Apress Media, LLC, all rights reserved. Use of this Work other than as provided for in this license is prohibited. By exercising any of the rights herein, you are accepting the terms of this license. You have the non-exclusive right to copy, use and distribute this English language Work in its entirety, electronically without modification except for those modifications necessary for formatting on specific devices, for all non-commercial purposes, in all media and formats known now or hereafter. While the advice and information in this Work are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

ISBN-13 (pbk): 978-1-4302-6637-2

ISBN-13 (electronic): 978-1-4302-6638-9

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director: Welmoed Spahr
Lead Editors: Steve Weiss (Apress); Patrick Hauke (Intel)
Coordinating Editor: Kevin Walter
Development Editor: Michael Koch
Cover Designer: Crest Premedia

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

www.dbooks.org

About ApressOpen

What Is ApressOpen?

- ApressOpen is an open access book program that publishes high-quality technical and business information.
- ApressOpen eBooks are available for global, free, noncommercial use.
- ApressOpen eBooks are available in PDF, ePub, and Mobi formats.
- The user friendly ApressOpen free eBook license is presented on the copyright page of this book.

Contents

About the Authors xv

About the Technical Reviewers xvii

Contributing Authors xix

Acknowledgments xxi

Chapter 1: Why Data Center Efficiency Matters 1

An Industry's Call to Action 2

Data Center Infrastructure Energy Use 3

Energy Proportional Server Efficiency 5

Regulatory Environment 6

Measuring Energy Efficiency 8

SPECPower 8

High Performance Computing Efficiency 14

Energy Efficiency and Cost 16

Summary 20

Chapter 2: CPU Power Management 21

Server CPU Architecture/Design 21

CPU Architecture Building Blocks 22

CPU Design Building Blocks 31

Intel Server Processors 32

www.dbooks.org

Introduction to Power 33
CPU Power Breakdown 34
Frequency, Voltage, and Temperature Interactions 36
Power-Saving Techniques 38
Turn It Off 38
Turn It Down 39
Power-Saving Strategies 39
CPU Power and Performance States 41
C-States 43
P-States 49
T-States 56
S-States and G-States 57
S0ix 57
Running Average Power Limit (RAPL) 58
CPU Thermal Management 66
CPU Power Management Summary 69
Summary 70
Chapter 3: Memory and I/O Power Management 71
System Memory 71
Memory Architecture Basics 71
Devices and Ranks 72
Memory Error Correction (ECC) 74
Memory Capacity 74
Device Power Characteristics 75
DDR3 vs. DDR4 76
RDIMMs, UDIMMs, SODIMMs, and LRDIMMs 77
Memory Channel Interleave and Imbalanced Memory Configurations 78

Power and Performance States 79
CKE Power Savings 79
Self-Refresh 81
Voltage/Frequency 82
DDR Thermal Management 83
Monitoring Temperature 83
Memory Throttling 83
CPU DDRIO 84
Workload Behavior 85
Memory Reliability Features 85
CPU I/Os 85
CPU Interconnect (QPI) 85
PCIe 87
Summary 90
Chapter 4: Platform Power Management 93
Platform Overview 93
Common Platform Components 93
Integration 95
Platform Manageability 97
CPU Sockets 97
Node Controllers 98
Memory Risers and Memory Buffer Chips 99
Server Chipsets 100
PCH and Platform Power Management 102
PCH Power Management 104
PCIe in Chipsets 105
PCH Thermal Management 105

www.dbooks.org

Networking 105
Ambient Temperature, TDP, and Thermal Management 106
Attached Media 108
LAN Power Management Features 109
USB 111
Link Power States 111
Link Frequency/Voltage 112
Storage 112
Storage Servers and Power Management 113
HDDs and SDDs 114
SATA and SAS Drive Power Management 114
Frequency/Voltage 116
NVMe Drive Power Management 117
Power Delivery 118
Overview of Power Delivery 118
Power Converter Basics 122
Power Conversion Losses 128
Thermal Management 137
System Considerations 141
Component Thermal Management Features 143
Platform Thermal Management 145
Fan Speed Control and Design 149
Summary 151
Chapter 5: BIOS and Management Firmware 153
BIOS Firmware 154
Microcode Update 155
Advanced Configuration and Power Interface 156
Setup Utility 162

Management Firmware 163
Node Manager Capabilities 163
IPMI 168
ACPI Power Metering Objects 171
Summary 171
Chapter 6: Operating Systems 173
Operating Systems 174
C-state Control 174
C-state Policy 176
P-state Control 178
P-state Policy 183
T-state Control 187
Global Power Policy 187
Process Scheduling 188
Memory Management 192
Device Drivers 193
Virtualization 195
Power State Control 195
Consolidation 198
VM Migration 199
Comparison of Operating Environments 201
Microsoft Windows Server (including Hyper-V) 201
Linux Distributions (including KVM) 203
VMWare ESX 206
Summary 207

www.dbooks.org

Chapter 7: Monitoring 209

Hardware Monitoring 209

Fixed Counters 210

Core Performance Monitors 210

Uncore Performance Monitors 211

Status Snapshots 213

Counter Access and Counter Constraints 214

Events and Metrics 214

Management Controller Monitoring 230

Component Power Sensors 230

Synthetic Sensors 231

Sensors and Events 231

Software Monitoring 235

Utilization and Processor Time 236

Processor Power State Requests 240

Scheduler, Processes, and Threads 243

Interrupts 244

Memory 245

I/O 247

Tools 249

Health Checks 249

Hardware Monitoring Tools 254

Operating System Monitoring Tools 265

Summary 268

Chapter 8: Characterization and Optimization 269

Workloads 271

Identifying Suitable Workloads 272

Workload Types 273

System Characterization 277

Steady State vs. Non-Steady State 277

Data Collection 278

Methodology 280

Analysis 281

Optimization 285

CPU Power Management 285

Memory 298

NIC 300

Storage 301

Thermal Management 302

Optimization at a Glance 303

Summary 306

Chapter 9: Data Center Management 307

Data Center Management and Power Distribution 307

Data Center Facilities 307

Power Infrastructure 308

Cooling Infrastructure 312

Simplified Total Cost Models of Cost and Compute Infrastructure 314

Performance per Watt per Dollar 316

Summary 317

Appendix A: Technology and Terms 319

Index 327

www.dbooks.org

About the Authors



Corey Gough is a principal engineer focused on server energy efficiency in Intel's Data Center Group. He currently leads efforts in power and performance analysis, system optimization, and new technology development with over 17 years of expertise in power/performance. Corey lives in Portland, Oregon, and earned his BS in Computer Science from the University of Oregon.

Winston Saunders has worked at Intel for nearly two decades and currently leads Security Technology Execution Initiatives in the Data Center Group there. Winston is a graduate of the University of California, Berkeley (UC Berkeley), and the University of Washington. You can follow him online @WinstonOnEnergy on Twitter.

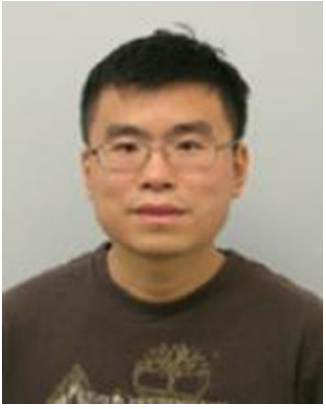


Ian Steiner has worked at Intel on the CPU architecture team for nearly a decade and is currently a member of the Product Development Group. His primary focus is on power management and power/performance optimization across the server product line. Ian lives in Portland, Oregon, and earned his MS in Electrical Engineering from the University of Illinois at Urbana-Champaign.

About the Technical Reviewers



Jonathan Koomey is a research fellow at the Steyer-Taylor Center for Energy Policy and Finance at Stanford University, has worked for more than two decades at Lawrence Berkeley National Laboratory, and has been a visiting professor at Stanford University (2003-4 and Fall 2008), Yale University (Fall 2009), and UC Berkeley's Energy and Resources Group (Fall 2011). He was a lecturer in management at Stanford's Graduate School of Business in Spring 2013. Dr. Koomey holds MS and PhD degrees from the Energy and Resources Group at UC Berkeley, and an AB in History of Science from Harvard University. He is the author or coauthor of 9 books and more than 200 articles and reports. He's also one of the leading international experts on the economics of reducing greenhouse gas emissions, the effects of information technology on resource use, and the energy use and economics of data centers. He's the author of *Turning Numbers into Knowledge: Mastering the Art of Problem Solving* (which has been translated into Chinese and Italian) and *Cold Cash, Cool Climate: Science-Based Advice for Ecological Entrepreneurs* (both from `Analytics Press`).



Liqun Cheng is a performance lead at Google, where he does large scale computing platform design, analysis, and tuning. His interests range from distributed system software infrastructure to energy proportional computing. He received his BS degree from Shanghai Jiao Tong University and his PhD degree from the University of Utah.

www.dbooks.org

Contributing Authors

Brian Griffith has been a power delivery engineer with Intel for 20 years working in the area of server power delivery and power management. Over this time period, he has developed power delivery solutions for multiple generations of server systems, has published requirements for server power supplies, and has been representing Intel in the PMBus forum. He has an MS degree in Electrical Engineering from Illinois Institute of Technology.

Andi Kleen is a software engineer in Intel's Open Source Technology Center. He is a long term Linux kernel contributor. He currently works on scalability to many cores and performance analysis.

Pankaj Kumar is the chief storage architect and a principal engineer with the Intel Storage Group, part of Intel's Data Center Group. Pankaj is responsible for the storage architecture vision, the future storage technology direction, and the platform storage extension roadmap. He has over 23 years of experience in the areas of storage, networking industry, and CPU/SoC design and architecture. Mr. Kumar holds a BS degree in Electronical Engineering and Communication from the National Institute of Technology Hamirpur (HP), India.

Eric Mann has been with the Intel Networking Division for over 17 years. He has contributed to various energy efficiency industry standards and regulatory bodies such as IEEE, Energy Star, and ECMA. He is a graduate of the University of Illinois-Urbana Champaign (94) with a BS in Computer Science.

Mariusz Oriol has been with Intel since 1999 and is currently working on manageability firmware for servers. His focus is on Node Manager power capping technology, including component enabling of operating systems, PSU, VR, and hot swap controllers. Mariusz received his MS in Software Engineering from Gdańsk University of Technology from the Faculty Electronics, Telecommunications and Informatics (ETI). He started his career at CrossComm (Boston, MA) in the division designing multiprotocol routers to support heterogeneous bridging and multiple routing protocols for computer networks. Mariusz lives and works in Poland.

Robin Steinbrecher is a server and data center thermal architect in Intel's Data Center Group. He is responsible for cooling architecture for server products including thermal management, cooling capability, and power/thermal optimization. Robin has over 30 years of experience in electronics cooling technologies at Intel and IBM, and now he focuses on integrating these technologies in data center applications.

Malay Trivedi is a silicon hardware architect at Intel specializing in Server PCH architecture. Prior to that, he worked on high frequency power conversion and power management. He earned his PhD from the University of Illinois, and has 20 patents.

Thomas Willhalm is a software engineer in Intel's Developer Relations Division. In this role, he helps independent software vendors optimize their software for Intel hardware and take advantage of its latest features. He is also one of the authors of *Intel Performance Counter Monitor* documentation.

Acknowledgments

We would like to thank technical and content reviewers who significantly improved the quality of the book. This includes **Ameya Ambardekar**, **Avinash Ananthakrishnan**, **Len Brown**, **Bill Carter**, **Gaurav Khanna**, **Naren Meadem**, **Kuljit Bains**, **Raghunathan Srinivasan**, **Ankush Varma**, **Vish Viswanathan**, **Sujal Vora**, **Brad Whyms**, and **Henry Wong**.

We would also like to thank **Dan Kingsley**, **Harry Li**, **Yingqi (Lucy) Lu**, and **Shubin Zhao** for providing the measurement data from data center workloads we used to illustrate concepts discussed in the book.

Wes Perdue and **Knut Grimsrud** provided key insights into HDD and SSD power management characteristics and trends.

Thanks to our editors and the people at Apress for their support and patience.

Finally, the authors would like to thank our families for their support while the authors spent nights, weekends, and vacations to make this book happen.

www.dbooks.org