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.

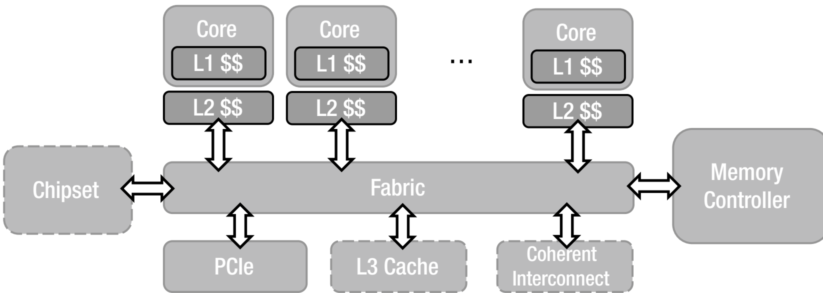

Figure 2-1. *A typical server CPU architecture block diagram*

[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.

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.

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.

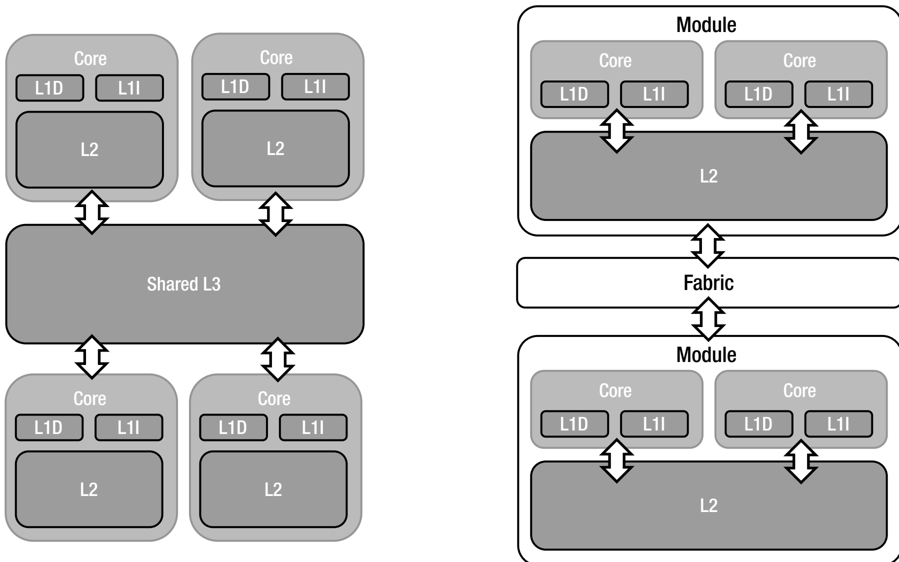

Figure 2-2. *Cache hierarchy examples*

[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.

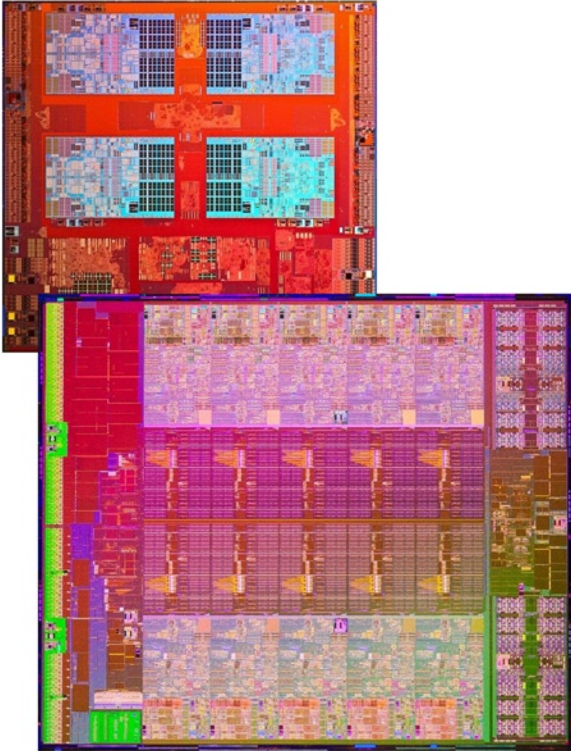

Figure 2-3. *Die photos of the 8c Atom Avoton (top) and 10c Xeon Ivy Bridge EP (bottom) (not to scale)*

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.

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.

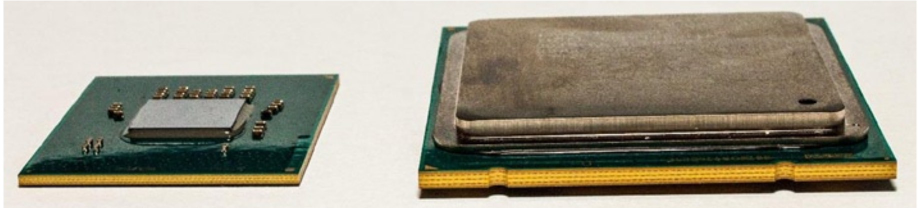

Figure 2-4. *Package photos of an 8c Xeon Sandy Bridge EP (right) and 8c Atom Avoton (left)*

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.

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.

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.

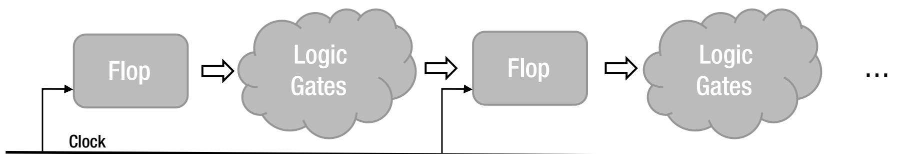

Figure 2-5. *Digital synchronous logic*

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

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.

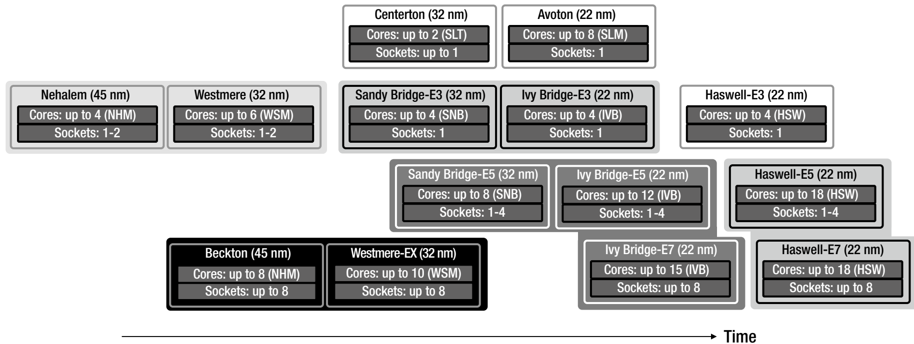

Figure 2-6. *Intel server processor progression*

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.

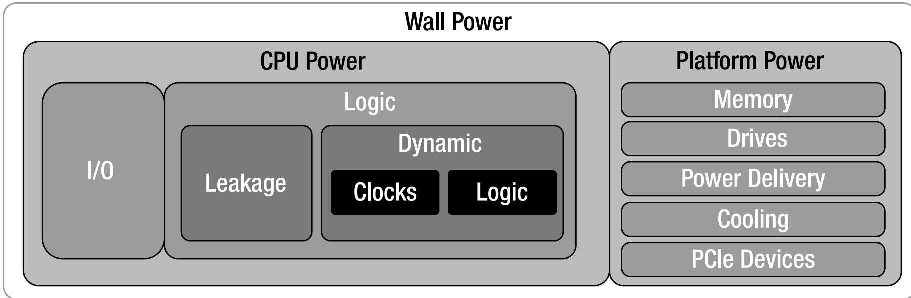

Figure 2-7. *Wall power breakdown illustration*

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.

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.

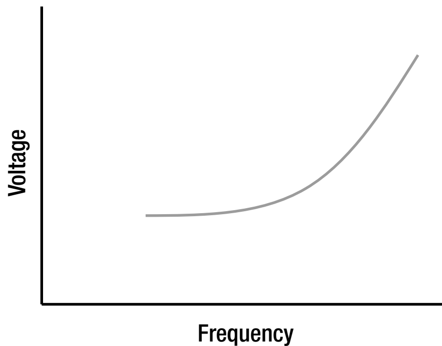

Figure 2-8. *Voltage/frequency relationships*

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.

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.

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 $<40\ \mu s$ 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*)

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*)

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.

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.

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.

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.

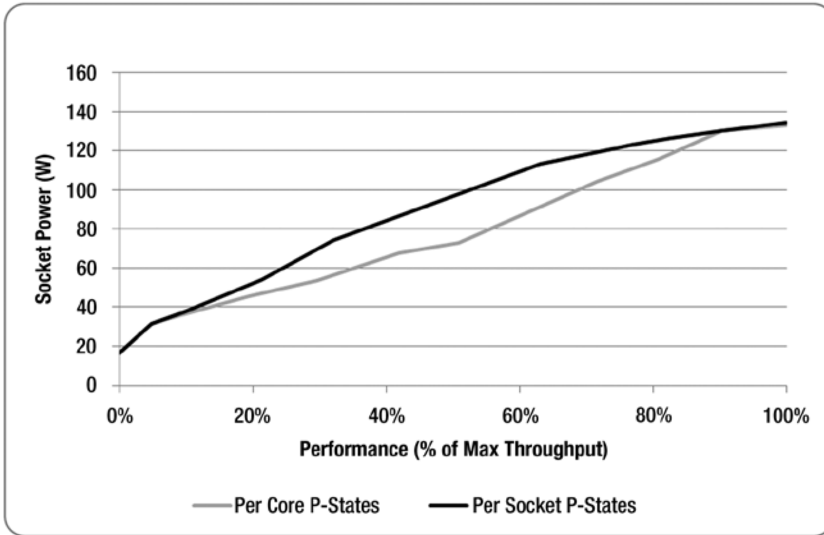

Figure 2-9. *Per core P-states (PCPS) vs. per socket P-states (PSPS)*

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.

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.

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.

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

S0*ix* 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 S0*ix* (today from S0i1 to S0i3) that provide successively deeper levels of power savings with increasing exit latencies. The S0*ix* 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.

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.

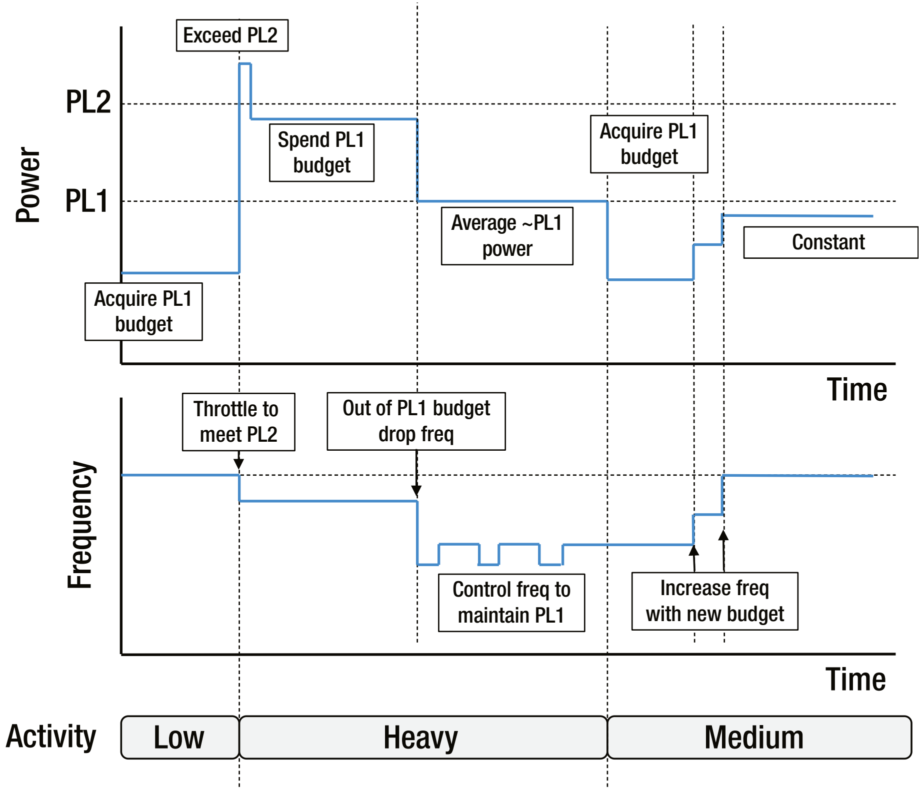

Figure 2-10. *Illustration of power-throttling with Turbo 2.0*

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

■ **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.

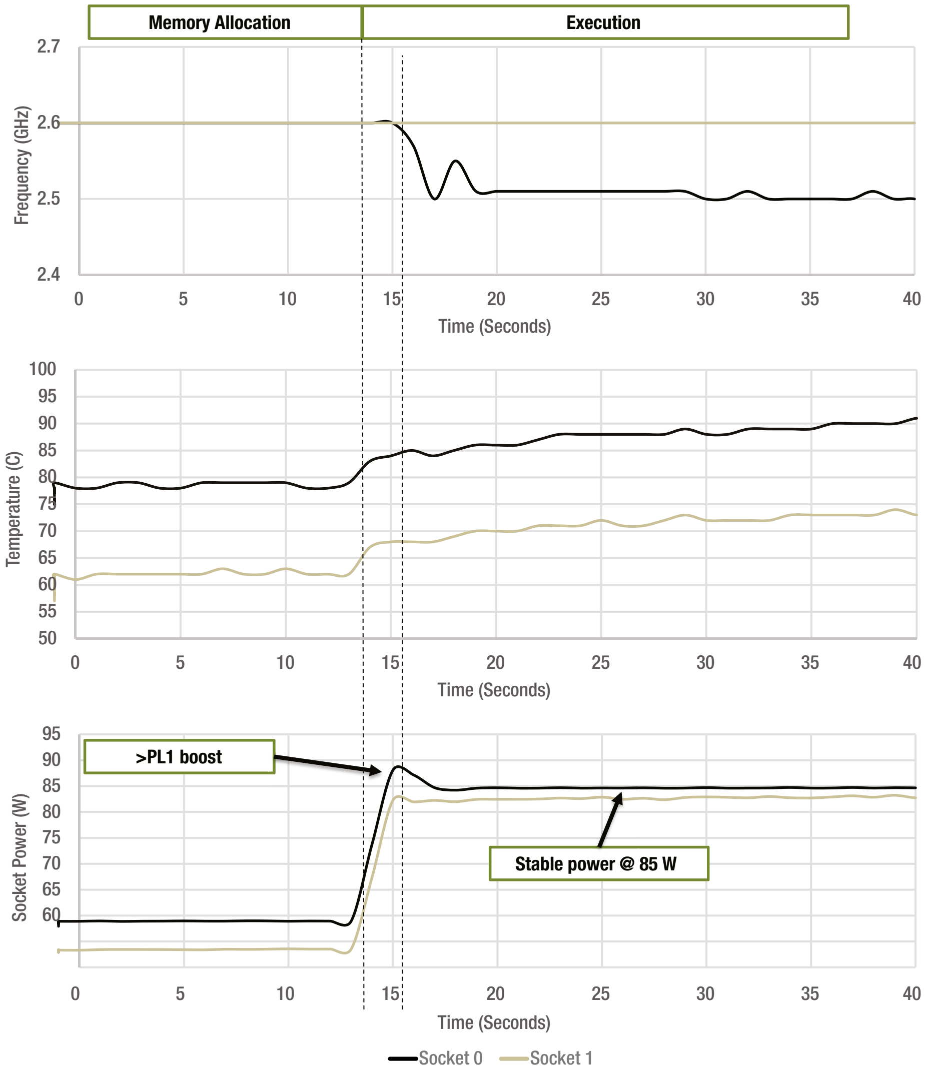

Figure 2-11. *Linpack power, frequency, and temperature with 85 W RAPL limit*

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

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

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.

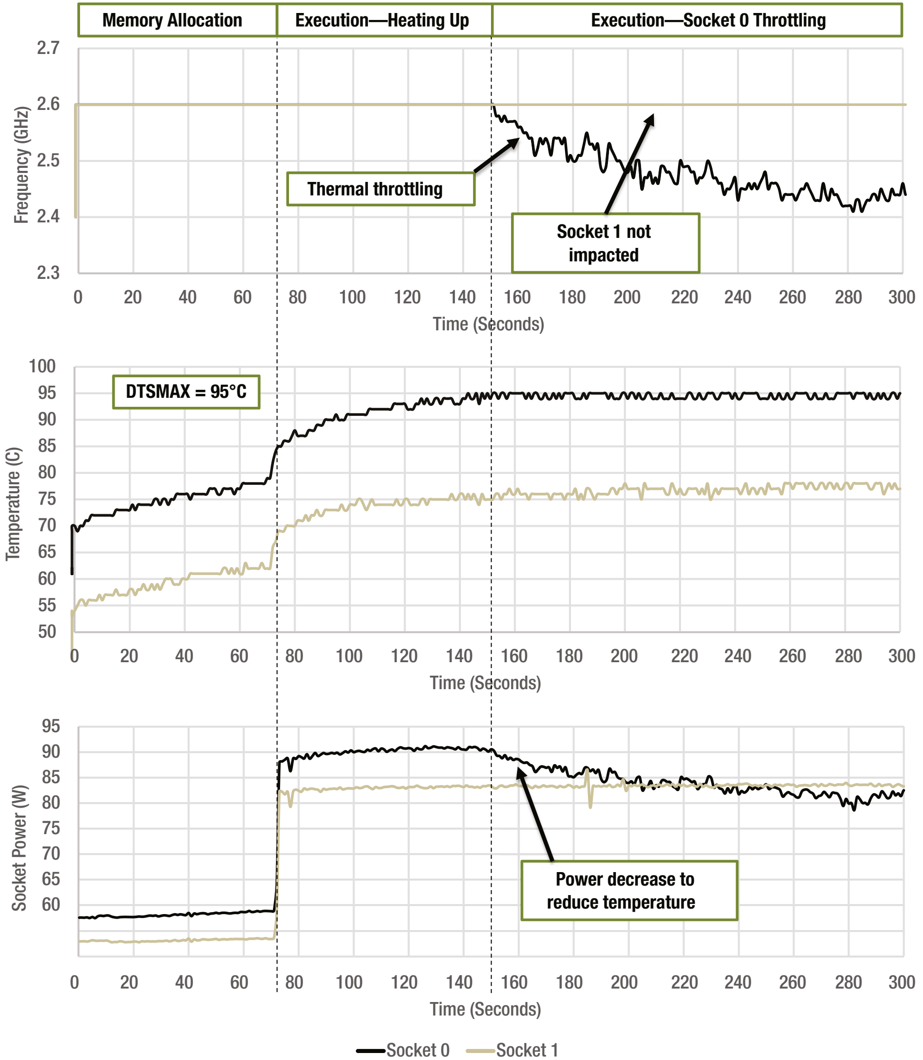

Figure 2-12. *Linpack frequency, temperature, and power when thermally throttled*

■ **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."

State		Turn It Off		Turn It Down		Resources			
		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.