ARTICLE TITLE: POWER MANAGEMENT STATES: P-STATES, C-STATES, AND PACKAGE C-STATES

Contents

Preface: What, Why and from Where

This is an aggregate of a series of blogs I wrote on power management states. That series is part of an even larger collection of blogs addressing all sorts of power management topics including power management states (this one), turbo / hyper-threading power states, power management configuration, and power management policy. With one exception, the discussion in these blogs should be generally useful to everyone, even though they concentrate on the Intel® Xeon Phi™ coprocessor. The exception is the series on configuration, which by its nature, is a more platform dependent topic and focused on the Intel® Xeon Phi™ coprocessor and the Intel® Manycore Platform Software Stack (MPSS). In addition to this collection of power management blogs, there are two other companion collections: a series on different techniques for measuring power performance[i], and another loose set of my earlier blogs that cover a variety of topics such as where $C*V^2*f$ comes from.

The article you are currently reading was originally published as the following series of 5 somewhat inconsistently entitled blogs:

- Intel Xeon Phi coprocessor Power Management Pt 0: Introduction and inquiring minds

- Intel Xeon Phi coprocessor Power Management Part 1: P-States, Reducing power consumption without impacting performance
- Intel Xeon Phi coprocessor Power Management Part 2a: Core C-States, The Details
- Intel Xeon Phi coprocessor Power Management Part 2b: Package C-States, The Details
- Intel Xeon Phi coprocessor Power Management Part 3: An Intuitive Description of Power States Using Stick Figures and Light Bulbs

In addition to these, there is a bonus article in an appendix:

- C-States, P-States, where the heck are those T-States?

At the Intel® Developer Zone, you can find the individual blogs listed in yet another article, List of Useful Power and Power Management Articles, Blogs and References.

So kick your feet up, lean back, and enjoy this look at the ever fascinating topic of power management.

Chapter 1: Introduction and Inquiring Minds

So exactly which power states exist on the Intel Xeon Phi coprocessor? What happens in each of the power states? Inquiring minds want to know. And since you are, no doubt, aggressively involved in high performance computing (HPC), I am sure you want to know also.

This is not going to be a high powered, in depth, up-to-your-neck-in-technical-detail type of treatise on power management (PM). If you want that, I suggest you read the Intel Xeon Phi Coprocessor Software Developer's Guide (SDG)[ii]. As a word of warning, when the Power Management section of the SDG refers to writers of SW (i.e. programmers), whether explicitly or implicitly, they do not refer to you or me. Its target audience consists of those poor lost souls that design operating systems (OSs) and drivers. (By the way, in an earlier life, I was one of those "poor lost souls.") One of the objectives of the series of blogs you are now reading is to look at PM from the perspective of an application developer, i.e. you or I, and not a writer of operating systems or drivers.

I am also not going to talk to what C, P and PC-states are. If you want an introduction to these concepts before digging into this blog series, I recommend (humbly) an earlier blog series I wrote on just that topic. See http://software.intel.com/en-us/user/266847/track. It is a little hard to separate the relevant power management blogs from all my other forum posts and videos, so I list the most important ones in this endnote[iii].

Briefly, the coprocessor has package-based P-states, core-based C-states (which are sometimes referred to as CC-states) and package-based C-states (PC-states). It also has the capability to operate in Turbo mode[3]. There are no per-core based P-states.

The host and coprocessor share responsibility for power management on the coprocessor. For some PM activities, the coprocessor operates alone. For others, the host component acts as a gate keeper, sometimes controlling PM, and at other times overriding actions taken by the coprocessor's PM.

In the forthcoming series, I discuss package-based P-states (including Turbo mode[iv]), core-based C-states and package-based PC-states. I will also discuss what control you have as an application developer over coprocessor PM.

Here is one last note. I cannot guarantee that all the Intel Xeon Phi coprocessor SKUs (i.e. coprocessor types) expose these power management features.

Chapter 2: P-States: Reducing Power Consumption without Impacting Performance

Right up front, I am going to tell you that P-states are irrelevant, meaning they will not impact the performance of your HPC application. Nevertheless, they are important to your application in a more roundabout way. Since most of you belong to a group of untrusting and always questioning skeptics (i.e. engineers and scientists), I am going to go through the unnecessary exercise of justifying my claim.

The P-states are voltage-frequency pairs that set the speed and power consumption of the coprocessor. When the operating voltage of the processor is lower, so is the power consumption. (One of my earlier blogs explains why at a high though technical level.) Since the frequency is lowered in tandem with the voltage, the lower frequency results in slower computation. I can see the thought bubble appearing over your head: "Under what situations would I ever want to enable P-states and, so, possibly reduce the performance of my HPC app?" Having P-states is less important in the HPC domain than in less computationally intense environments, such as client-based computing and data-bound servers. But even in the coprocessor's HPC environment, longer quiescent states are common between large computational tasks. For example, if you are using the offload model, the coprocessor is likely unused during the periods between offloads. Also, a native application executing on the coprocessor will often be in a quiescent phase for many reasons, such as their having to wait for the next chunk of data to process.

The P-states the coprocessor Power Management (PM) supports is P0 through P*n*. The number of P-states a given coprocessor SKU (i.e. type) supports will vary, but there are always at least two. Similarly, some SKUs will support turbo P-states. The coprocessor's PM SW handles the transition from one P-state to another. The host's PM SW has little if any involvement.

A natural thing to wonder is how much this is going to impact performance; we do happen to be working in an HPC environment. The simple answer is that there is, in any practical sense, no impact on HPC performance. I am sure that, at this point, you are asking yourself a series of important questions:

(a) "Wait a minute; how can this be? If the coprocessor is slowing down the processor by reducing the frequency, how can this not affect the performance of my application?"

(b) "I just want my application to run as fast as possible. Why would I want to reduce power consumption at all?"

Let us first consider (b). I understand that power is not a direct priority for you as an application writer. Even so, it does impact your application's performance indirectly. More power consumption has to do with all that distant stuff, like higher facility costs in terms of electricity usage due to greater air-conditioning demands, facility space needs, etc. It is simply part of that unimportant stuff administrators, system architects, and facilities management worry about.

Truth be told, you need to worry about it also. This does impact your application in a very important way, though how is not initially obvious. If the facility can get power consumption down while not losing performance, this means that it can pack more processors in the same amount of space all while using the same power budget. To use another American idiom, you get more "bang for the buck." And this is a good thing for you as a programmer/scientist. When you get right down to it, lower power requirements mean that you can have more processors running in a smaller space, which means that you, as an application designer/scientist, can run not only bigger problems (more cores) but run them faster (less communication latency between those more cores).

Let us go back to P-states. P-states will impact performance in a theoretical sense, but not in a way that is relevant to an HPC application. How is this possible? It is because of how transitions between the P-states happen. It all has to do with processor utilization. The PM SW periodically monitors the processor utilization. If that utilization is less than a certain threshold, it increases the P-state, that is, it enters the next higher power efficiency state. The word term in the previous sentence is "utilization". When executing your computationally intensive HPC task on the coprocessor, what do you want your utilization to be? Ideally, you want it to be as close to 100% as you can get. Given this maximal utilization, what do you imagine is the P-state your app is executing in? Well, it is P0, the fastest P-state (ignoring turbo mode). Ergo, the more energy efficient P-states are irrelevant to your application since there is almost never a situation where a processor supporting your well-tuned HPC app will enter one of them.

So in summary, the "HPC" portions of an HPC application will run near 100% utilization. Near 100% utilization pretty much guarantees always using the fastest (non-turbo) P-state, P0. Ergo, P-states have essentially no impact upon the performance of an HPC application.

How do I get my application to run in one of those turbo modes? You cannot as it is just too dangerous. It is too easy to make a minor mistake resulting in overheating and damaging the coprocessor. If your processor supports turbo, leave its management to the OS.

Chapter 3: Core C-States - The Details

BACKGROUND: A QUICK REFRESHER ON IDLE STATES

Here is a quick summary of what C-states are. C-states are idle power saving states, in contrast to P-states, which are execution power saving states. During a P-state, the processor is still executing instructions, whereas during a C-state (other than C0), the processor is idle, meaning that nothing is executing. To make a quick analogy, a processor lying idle is like a house with all the lights on when no one is at home. Consuming all that power is doing nothing other than providing your electric company a little extra income. What is the best option? If no one is at home, meaning the house is idle, why leave the lights on? The same applies to a processor. If no one is using it, why keep the unused circuits powered up and consuming energy? Shut them down and save.

C0 is the “null” idle power state, meaning it is the non-idle state when the core is actually executing and not idle.

THE DIFFERENCE BETWEEN CORE AND PACKAGE IDLE STATES

The coprocessor has up to 60+ cores in one package. Core idle power states (C-states) are per core, meaning that one of those 60+ cores can be in C0, i.e. it is executing and not idle, while the one right next door is in a deep power conservation state of C6. In contrast, PC-states are Package idle states which are idle power conservation states for the entire package, meaning all 60+ cores and supporting circuitry on the silicon. As you can guess, to drop the package into a PC-6 state, all the cores must also be in a C6 state. Why? Since the package has functionality that supports all the cores, to “turn off” some package circuitry impacts all of them.

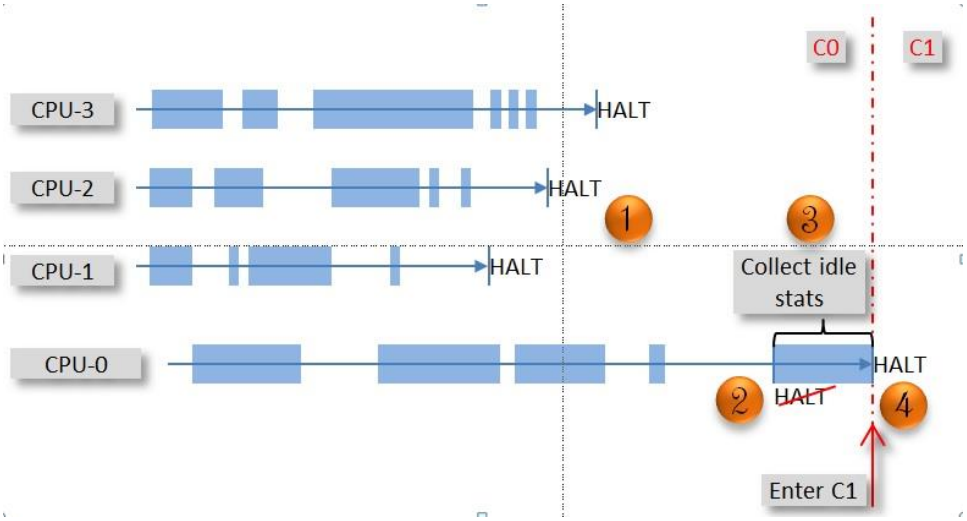

Figure 1. Dropping a core into a core-C1 state

WHAT CORE IDLE STATES ARE THERE?

Each core has 2 idle states, C1 and C6 (and, of course, C0).

C0 to Core-C1 Transition: Look at Figure 1. C1 happens when all 4 hardware (HW) threads supported by a core have executed a HALT instruction. At this point, let us now think of each of the 4 HW threads as the Operating System (OS) perceives them, namely as 4 separate CPUs (CPU 0 through 3). Step 1: the first three CPUs belonging to that core execute their HALT instruction. Step 2: that last CPU, CPU-0 in the illustration, attempts to execute its HALT instruction. Step 3: it interrupts to an idle residency data collection routine. This routine collects, you guessed it, idle residency data and stores that data in a data structure accessible to the OS. CPU 0 then HALTs. Step 4: at this point, all the CPUs are halted and the core enters a core-C1 state. In the core-C1 state, the core (and its "CPUs") is clock gated[v].

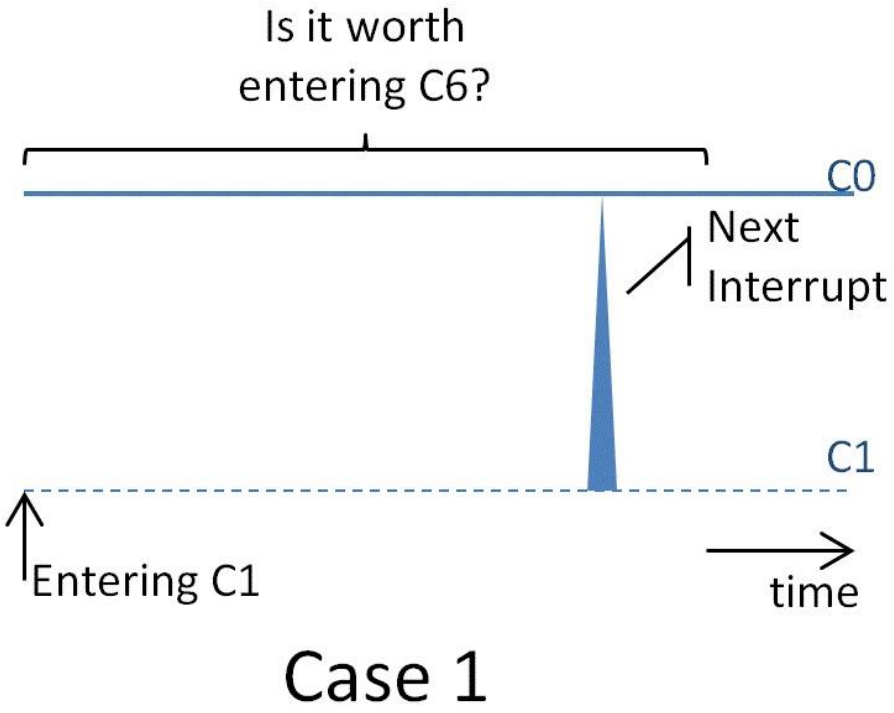

Figure 2. Is it worth entering C6: Is the next interrupted far enough out?

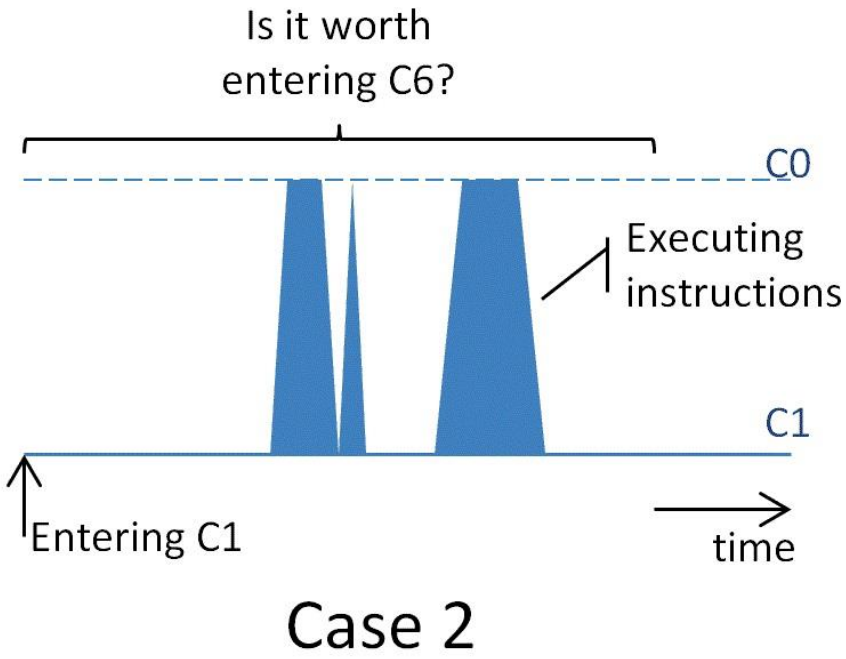

Figure 3. Is it worth entering C6: Is the estimated idle time high enough?

After entering core-C1: Now that the core is in C1, the coprocessor's Power Management routine comes into play. It needs to figure out whether it is worthwhile to shut the core down further and drop it into a core-C6 state. In a core-C6 state, further parts of the core are shut down and power gated. Remember that the coprocessor's Power Management SW executes on the OS core, typically core 0, and is not affected by the shutdown of other cores.

What type of decisions does the coprocessor's Power Management have to make? There are two primary ones as we discussed in the last chapter: Question#1: Will there (probably) be a net power savings? Question #2: Will any restart latency adversely affect the performance of the processor or of applications executing on the processor? Those decisions correspond to two major scenarios and are shown in Figures 2 and 3 above. Scenario 1 is where the coprocessor PM looks at how far away is the next scheduled or expected interrupt. If that interrupt is soon enough, it may not be worth shutting down the core further and suffering the added latency caused by bringing the core back up to C0. As is the case in life, the processor can never get anything for free. The price of dropping into a deeper C state is an added latency resulting from bringing the core/package back up to the non-idle state. Scenario 2 is where the coprocessor's Power Management looks at the history of core activity (meaning its HW threads) and figures out whether the execution (C0) and idle (C1) patterns of the core make core-C6 power savings worthwhile.

If the answers to both of these questions are "yes", then the core drops down into a core-C6 state.

After entering core-C6: Well dear reader, it looks like I have run out of time. The processor next decides if it can drop into the package idle states. I will cover that discussion in my next blog in this series.

Chapter 4: Package C-States - The Details

TERMINOLOGY NOTE:

Upon reading the SDG (Intel Xeon Phi Coprocessor Software Developer's Guide), you'll find a variety of confusing names and acronyms. Here's my decoder ring:

Package Auto C3[vi]: also referred to as Auto-C3, AutoC3, PC3, C3, Auto-PC3 and Package C3

Package Deep-C3: also referred to as PC3, DeepC3, DeeperC3, Deep PC3 and Package C3 (No, I am not repeating myself.)

Package C6: Also referred to as PC6 and C6 and Package C6.

BACKGROUND: WHAT THE HECK IS THE "UNCORE"?

Before we dig deep into package C-states, I want to give you some background about circuitry on a modern Intel® processor. A natural way of dividing up the circuitry of a processor is that composing the

cores -- basically that supporting the pipeline, ALUs, registers, cache, etc. -- and everything else (supporting circuitry). It turns out that “everything else” can be further divided into that support circuitry not directly related to performance (e.g. PCI Express* interfacing), and that which is (e.g. the bus connecting cores). Intel calls support circuitry that directly impacts the performance of an optimized application the “Uncore”.

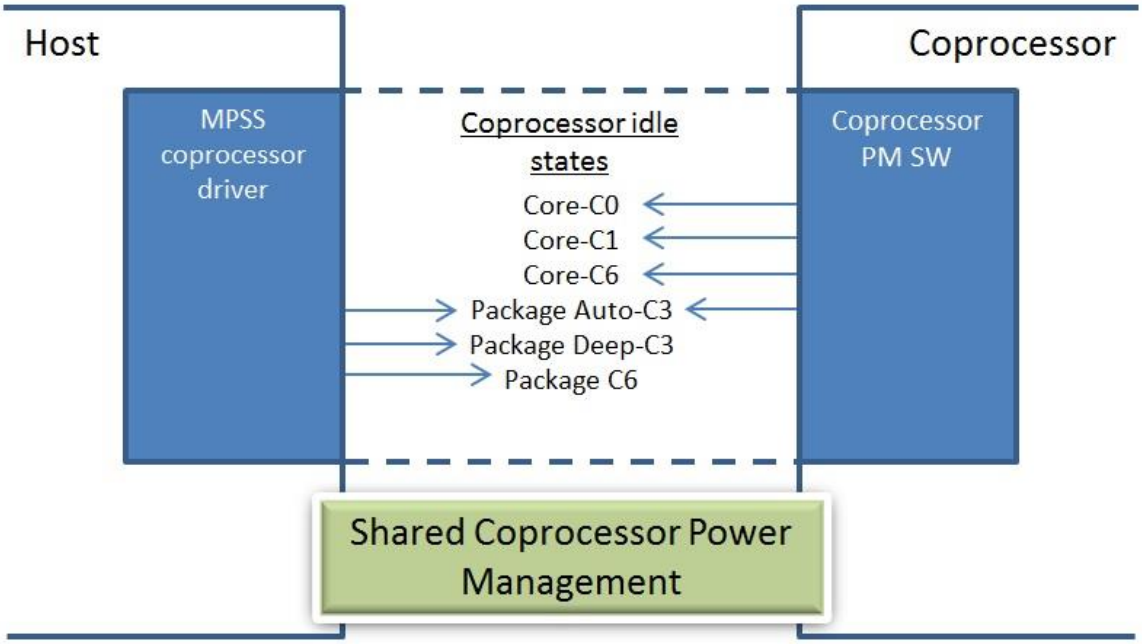

Figure 4. Circuitry types on the coprocessor

Since that is out of the way, let us get back to package C-states.

WHY DO WE NEED PACKAGE C-STATES?

After gating the clocks of every one of the cores, what other techniques can you use to get even more power savings? Here’s a trivial and admittedly flippant example of what you could do: unplug the processor. You’d be using no power, though the disadvantages of pulling the power plug are pretty obvious. A better idea is to selectively shutdown the more global components of the processor in such a way that you can bring the processor back up to a fully functional state (i.e. C0) relatively quickly.

Package C-States are just that - the progressive shutdown of additional circuitry to get even more savings. Since we have already shut down the entire package’s circuitry associated with the cores, the remaining circuitry is necessarily common to all the cores, thus the name “package” C-states.

WHAT PACKAGE IDLE STATES ARE THERE?

My dear readers, there are 3 package C states: Auto-C3, Deep-C3, and (package) C6. As a reminder, all these are package C-states, meaning that all the threads/CPUs in all the cores are in a HALT state. I know what you are thinking. “If all the cores in the coprocessor are in a HALT state, how can the Power Management (PM) software (SW) run?” That’s a good question. The answer is obvious once you think on it. If the PM SW can’t run on the coprocessor, where can it run? It runs on the host, of course.

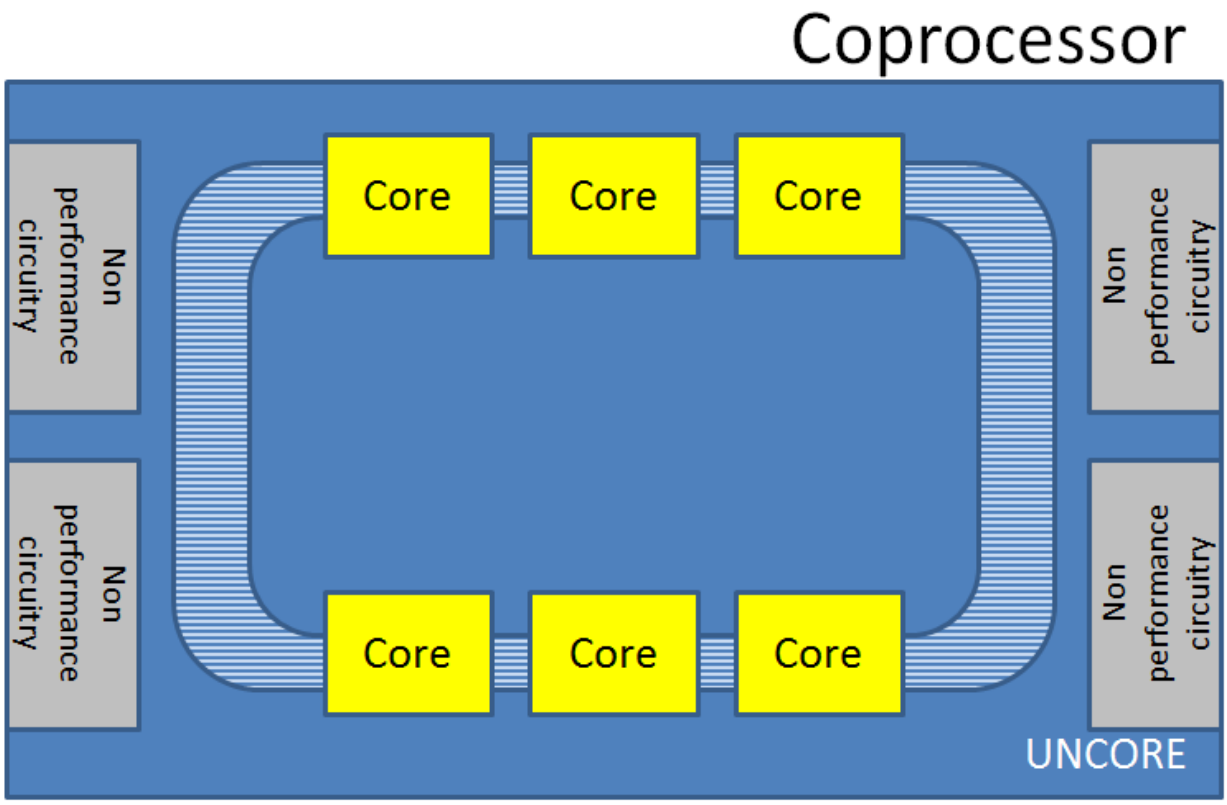

Figure 5. Coprocessor and host power management responsibilities and control

There are two parts to controlling power management on the Intel® Xeon Phi™ coprocessor, the PM SW that runs on the coprocessor, and the PM component of the MPSS Coprocessor Driver that runs on the host. See figure 1. The coprocessor part controls transitions into and out of the various core C-states. Naturally, when it is not possible for the PM SW to run on the coprocessor, such as for package Deep-C3 and package C6, the host takes over. Package Auto-C3 is shared by both.

WHAT IS SHUT DOWN IN THE PACKAGE C-STATES?

I was going to rewrite this table but it is so clear, I am stealing it instead. It is Table 3-2 of the Intel® Xeon Phi™ Coprocessor Software Developer’s Guide (SDG).

Package Idle State	Core State	Uncore State	TSC/LAPIC	C3WakeupTimer	PCI Express* Traffic
PC3	Preserved	Preserved	Frozen	On expiration, package exits PC3	Package exits PC3
Deep C3	Preserved	Preserved	Frozen	No effect	Times out
PC6	Lost	Lost	Reset	No effect	Times out

And for those of you who want a little more detail:

Package Auto-C3: Ring and Uncore clock gated

Package Deep-C3: VccP reduced

Package C6: VccP is off (I.e. Cores, Ring and Uncore are powered down)

TSC and LAPIC are clocks which stop when the Uncore is shutdown. They have to be set appropriately when the package is reactivated. “PC3” is the same as the package Auto-C3 state.

HOW IDLE PACKAGE C-STATE TRANSITIONS ARE DETERMINED

Into Package Auto-C3: You can think of the first package state, Auto-C3, as a transition state. The coprocessor PM SW can initiate a transition into this state. The MPSS PM SW can override this request under certain conditions, such as when the host knows that the Uncore part of the coprocessor is still busy.

We will also see that the package Auto-C3 state is the only package state that can be initiated by the coprocessor’s power management. Though this seems a little unfair at first, upon further thinking the reason is obvious. At the start of a transition into package Auto-C3, the coprocessor SW PM routine is running and can initiate the transition into the first package state. (To be technically accurate, the core executing the PM SW can transition quickly out of a core C-state into C0 quickly)

Beneath Auto-C3, the coprocessor isn’t executing and transitions to deeper package C-states are best controlled by the host PM SW. Not only is this due to the coprocessor’s own PM SW is essentially suspended, but because the host can see what is happening in a more global sense, such as Uncore activity after all the cores are gated, and traffic across the PCI Express bus.

Into Package Deep-C3: The host's coprocessor PM SW looks at idle residency history, interrupts (such as PCI* Express traffic), and the cost of waking the coprocessor up from package Deep-C3 to decide whether to transition the coprocessor from a package Auto-C3 state into a package Deep-C3 state.

Into Package C6: Same as the Package Deep-C3 transition but only more so.

Chapter 5: An Intuitive Description of Power States Using Stick Figures and Light Bulbs

AN INTUITIVE ILLUSTRATION OF A CORE AND ITS HW THREADS

This is the fourth installment of a series of blogs on Power Management for the Intel Xeon Phi coprocessor.

For those of you who have read my blog presenting an intuitive introduction to the Intel Xeon Phi coprocessor, The Intel Xeon Phi coprocessor: What is it and why should I care? PART 3: Splitting Hares and Tortoises too, I irreverently referred to "diligent high tech workers who labor ceaselessly for their corporate masters". Let's take this description a little further. In Figure A, we have one such diligent high tech worker. He is analogous to one coprocessor CPU/HW thread.

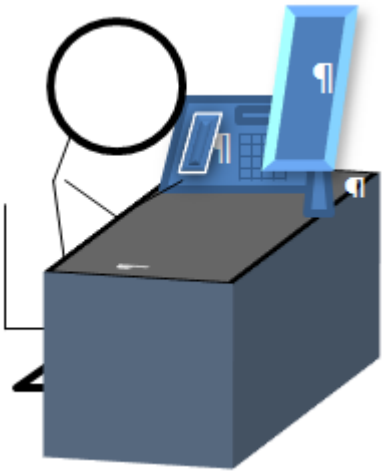

Figure 6. Diligent high tech worker, i.e. an Intel® Xeon Phi™ HW thread

There are 4 HW threads to a core. See Figure B. It's pretty obvious so I'm not going to bother with a multipage boring description of what it means. There is also that mysterious light bulb. The light bulb represents the infrastructure that supports the core, such as timing and power circuits.

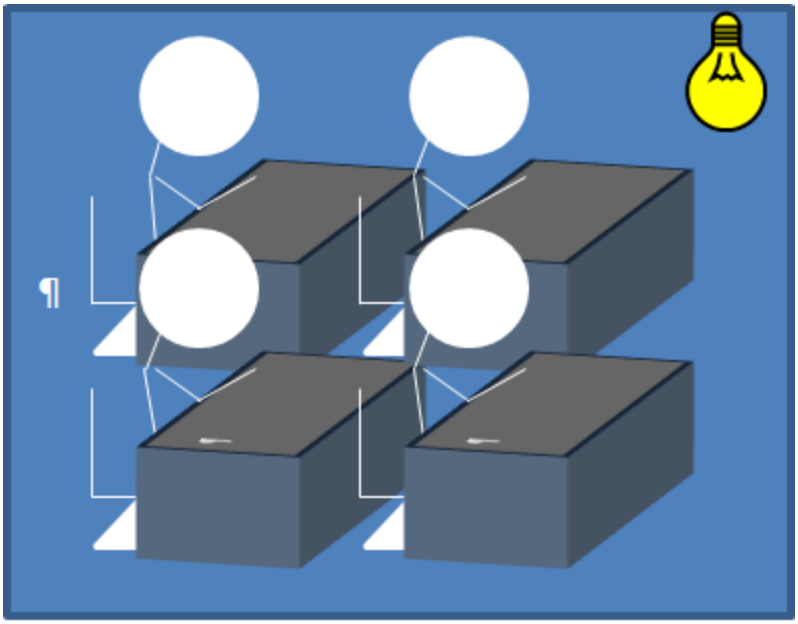

Figure 7. Diligent high tech workers in a room, i.e. an Intel® Xeon Phi™ coprocessor core

POWER MANAGEMENT: Core C0 and C1

So what does all this have to do with power management? Though it is sometimes assumed by the lower paid liberal arts students that engineers are unimaginative and boring, you and I know that, though boring we may be, we are not unimaginative. With this in mind, I ask you to visualize that on every one of those desks is a computer and a desk light.

The Core in C0: When at least one of the high tech workers is diligently working at their task. (I.e. At least one of the core's CPUs/HW threads is executing instructions.)

CPU Executing a HALT: When one of those diligent workers finishes his task, he turns out his desk lamp, shuts down his computer, and leaves. (I.e. one of the HW threads executes a HALT instruction.)

Entering Core-C1: When all four diligent workers finish their tasks, they all execute HALT instructions. The last one finishing turns off the lights. (I.e. The core is clock gated.)

POWER MANAGEMENT: Core-C6

Entering Core-C6: Yes, I know it's blatantly obvious, but I like talking to myself. As time proceeds, everyone leaves for lunch. Since no one is in the office, we can shut things down even further in the rooms (i.e. power gating). Remember, though, that they are coming back after lunch so anything shut down must be able to be powered back up quickly.

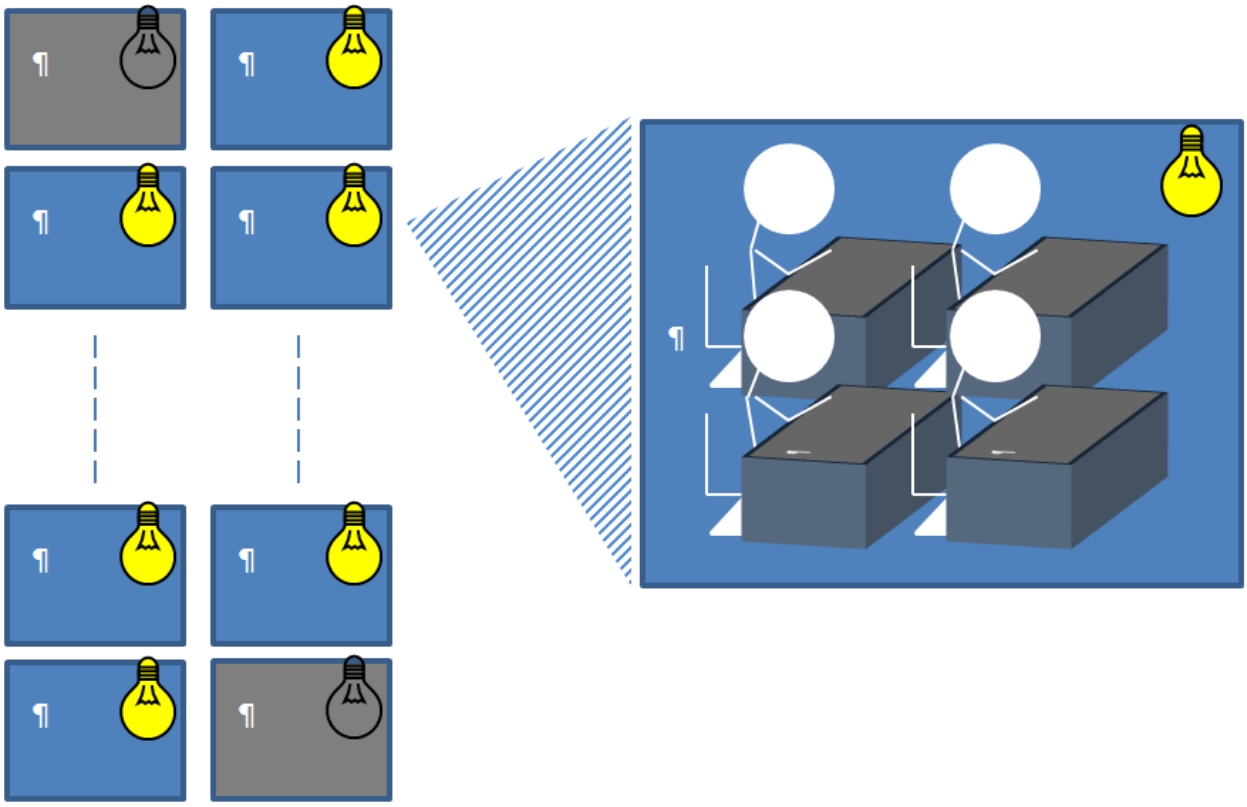

Figure 8. A building full of diligent high tech workers, i.e. an Intel® Xeon Phi™ coprocessor

POWER MANAGEMENT: Package Auto-C3, Package Deep-C3 and Package C6

Now I'm going to stretch this analogy a little bit, but since it is fun, I'm going to keep on going.

Let's expand this very creative analogy. Imagine if you will, a building with many rooms, 60+ in point of fact. See Figure C. Yes, I know that here in Silicon Valley, diligent high tech workers work in luxurious cubes, not stuffy offices. Unfortunately, the analogy breaks down at that point so I am sticking with communal offices.

Entering Package Auto-C3: Everyone has left the floor, so the movement sensor automatically shut off the floor lights. (I.e. the coprocessor power management software clock gates the Uncore and other support circuitry on the silicon).

Entering Package Deep-C3: It's the weekend so facilities (i.e. the MPSS Coprocessor Driver Power Management module) shuts down the air condition and phone services. (I.e. the host reduces the coprocessor's VccP and has it ignore interrupts.)

Entering Package C6: It's Christmas week shutdown and forced vacation time, so facilities turns off all electricity, air condition, phones, servers, elevators, toilets, etc. (I.e. the host turns off the coprocessor's Vccp and shuts down its monitoring of PCI Express* traffic.)

POWER MANAGEMENT: Getting Obsessive

Having fun with this analogy, I was thinking of extending it further into industrial campuses (a node containing multiple coprocessors), international engineering divisions (clusters with each node containing multiple coprocessors) and contracting with external partners (distributed WAN processing). Sanity and common sense prevailed and I leave the analogy as is.

Summary

We discussed the different types of power management states. Though the concepts are general, we concentrated on a specific platform, the Intel® Xeon Phi™ coprocessor. Most modern processors, be they Intel Corporation, AMD* or embedded, have such states with some variation.

Intel® processors have two types of power management states, P-states (runtime) and C-states (idle). The C-states are then divided into two more categories, core and package. P-states are runtime (C0) states and reduce power by slowing the processor down and reducing its voltage. C-states are idle states meaning that they shutdown parts of the processor when the cores are unused. There are two types of C-states. Core C-states shutdown parts of individual cores/CPUs. Since modern processors have multiple cores, package C-states shutdown the circuitry that supports those cores.

The net effect of these power states is to substantially reduce the power and energy usage of modern Intel® processors. This energy reduction can be considerable, in some cases by over an order of magnitude.

The impact of these power savings cannot be over stated for all platforms from smart phones to HPC clusters. For example, by reducing the power and energy consumption of the individual processors in an HPC cluster, the same facility can support more processors. This increases processor density, reduces communication times between nodes, and makes possible a much more powerful machine that can

address larger and more complex problems. At the opposite end in portable passively cooled devices such as smart phones and tablets, reduced power and energy usage lengthens battery life and reduces cooling issues. This allows more powerful processors which in turn increases the capability of such devices.

Appendix: C-States, P-States, where the heck are those T-States?

I had an interesting question come across my desk a few days ago: “Is it still worthwhile to understand T-states?” My first response was to think, “Huh? What the heck is a T-state?”

Doing a little more research, I discovered that, yes, there is something called a T-state, and no, it really isn’t relevant any more, at least for mainline Intel® processors.

Let me say this again: **T-States are no longer relevant!**

Now that the purely practical people have drifted off to other more “relevant” activities, here’s something for all you power management history buffs.

A T-state was once known as a Throttling state. Back in the days before C and P states, T-states existed to save processors from burning themselves up when things went very badly, such as when the cooling fan failed while the processor was running as fast as she could. If a simple well placed temperature sensor registered that the junction temperature was reaching a level that could cause damage to the package or its contents, the HW power manager would place the processor in different T-States depending upon temperature; the higher the temperature, the higher the T-State.

As you probably already guessed, the normal run state of the processor was T0. When the processor entered a higher T-state, the manager would clock gate the cores to slowdown execution and allow the processor to “relax” and cool. For example, in T1 the HW power manager might clock gate 12% of the cycles. In rough terms, this means that the core will run for 78% of the time and sleep for the rest. T2 might clock gate 25% of the cycles, etc. In the very highest T-state, over 90% of the cycles might be clock gated. (See the figure below.)

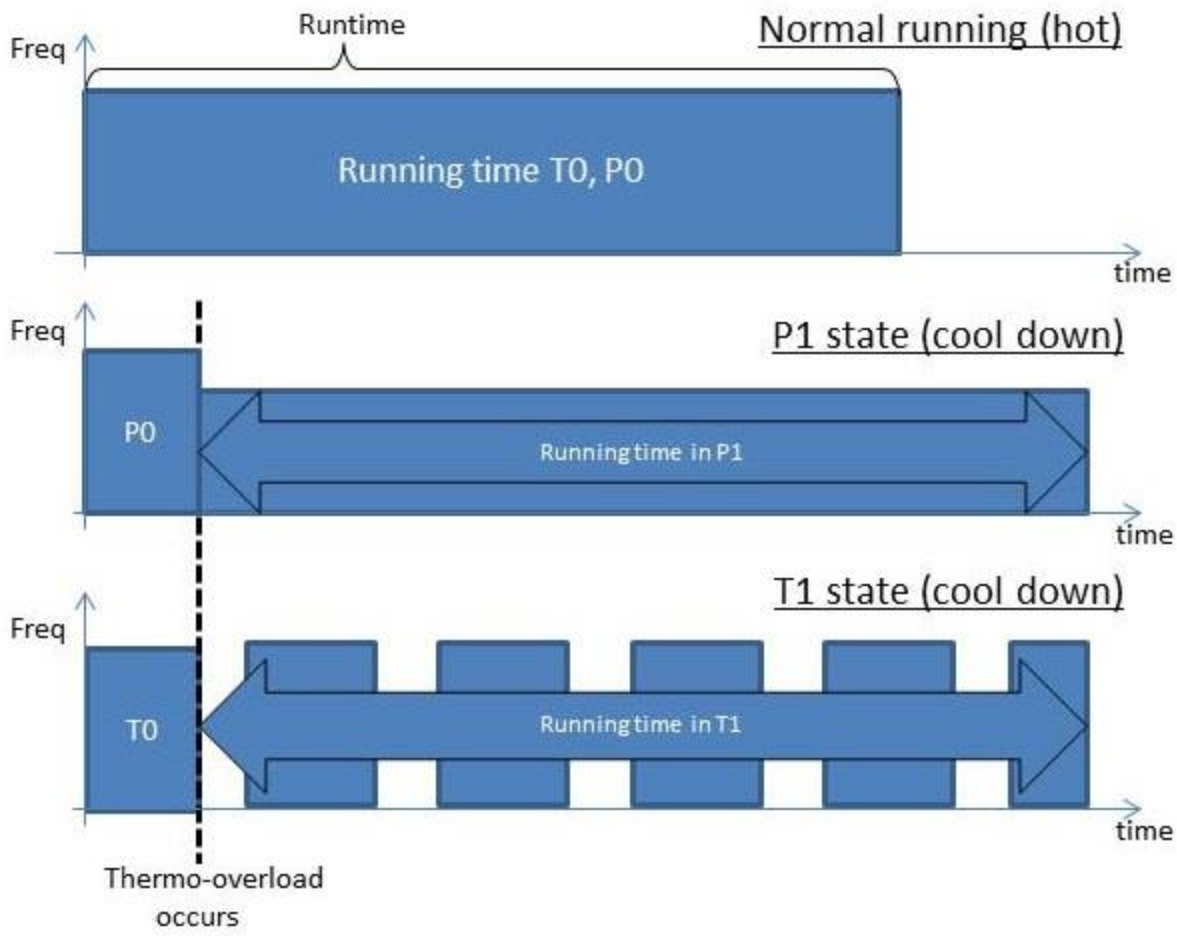

Figure 9. Running Time for T0/P0, P1, and T1 States

Note that in contrast to P-states, the voltage and frequency are not changed. Also, using T-states the application runs slower not because the processor is running slower, but because it is suspended for some percent of the time. In some ways, you can think of a T-state as being like a clock gated C1 state with the processor not being idle, i.e. it is still doing something useful.

In the figure above, the top most area shows the runtime of a compute intensive workload if no thermal overload occurs. The bottom shows the situation with T states (i.e. before P states), where the processor begins to toggle between running and stopped states to cool down the processor. The middle is what happens in current processors, where the frequency/voltage pair is reduced allowing the processor to cool.

For those of you who have borne with me for the history lesson, there are a few more practical reasons you should be at least aware of T-states.

(1) Some technical literature now uses the term “throttling states” to mean P-states, not T-states.

(2) Some power management data structures, such as some defined by ACPI, still include an unused T-state field. Many inquiries about T-states originate from this little fact.

(3) I suspect that T-states are still relevant in some embedded processors

References

Kidd, Taylor (10/23/13) - "List of Useful Power and Power Management Articles, Blogs and References," http://software.intel.com/en-us/articles/list-of-useful-power-and-power-management-articles-blogs-and-references, downloaded 3/24/2014.

For those of you with a passion for power management, check out the Intel Xeon Phi Coprocessor Software Developer's Guide. It has state diagrams and other goodies. I recommend sections 2.1.13, "Power Management", and all of section 3.1, "Power Management (PM)" for your late night reading.

NOTE: As previously in my blogs, any illustrations can be blamed solely on me as no copyright has been infringed or artistic ability shown.

Endnotes

[i] I expect to publish these blogs over the May / June 2014 time frame.
[ii] http://software.intel.com/en-us/articles/intel-xeon-phi-coprocessor-system-software-developers-guide
[iii] Here is a list of my most relevant earlier blogs
http://software.intel.com/en-us/blogs/2008/03/04/introduction-to-power-management-on-intel-processors
http://software.intel.com/en-us/blogs/2008/05/29/what-exactly-is-a-p-state-pt-1
http://software.intel.com/en-us/blogs/2008/03/27/update-c-states-c-states-and-even-more-c-states
http://software.intel.com/en-us/blogs/2008/04/29/theres-got-to-be-a-catch
http://software.intel.com/en-us/blogs/2008/05/29/what-exactly-is-a-p-state-pt-1
http://software.intel.com/en-us/blogs/2008/08/15/so-how-are-p-states-related-to-power-management
http://software.intel.com/en-us/blogs/2008/07/31/can-p-states-save-overall-energy
[iv] For those that need a quick refresher, turbo mode is a set of over-clocked P-states that exceed the normal power limits of the silicon. If normally run in this P-state, the silicon would over heat and potentially burn up. Turbo is possible because these normal power limits are computed based upon every core running at maximum performance. There are many situations where the entire power budget is not utilized. In these cases, the power management SW can allow a temporary overclocking.
[v] CPUs have at least one oscillator (clock) that emits a timing pulse. The circuits of the processor use this timing pulse to coordinate all activities.

[vi] Auto-PC3 may have been removed as of MPSS 3.1. Even so, it is worthy of a discussion as it illustrates the impact of latency and local vs remote management

About the Author

Taylor Kidd is an Intel application engineer. His interest is in high performance computing architectures and optimization. His background includes application development, research and teaching in the areas of embedded systems, distributed operating systems and computer architectures. He got his Ph.D. from UCSD and proudly points to his training as an electronic technician. He currently writes a blog on power management within Intel processors.

Notices

Performance Notice

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804