Proceedings of the Linux Symposium

Volume Two

July 20nd–23th, 2005
Ottawa, Ontario
Canada

Conference Organizers

Andrew J. Hutton, *Steamballoon, Inc.*
C. Craig Ross, *Linux Symposium*
Stephanie Donovan, *Linux Symposium*

Review Committee

Gerrit Huizenga, *IBM*
Matthew Wilcox, *HP*
Dirk Hohndel, *Intel*
Val Henson, *Sun Microsystems*
Jamal Hadi Salimi, *Znyx*
Matt Domsch, *Dell*
Andrew Hutton, *Steamballoon, Inc.*

Proceedings Formatting Team

John W. Lockhart, *Red Hat, Inc.*

Chip Multi Processing aware Linux Kernel Scheduler

Suresh Siddha
suresh.b.siddha@intel.com

Venkatesh Pallipadi
venkatesh.pallipadi@intel.com

Asit Mallick
asit.k.mallick@intel.com

Abstract

Recent advances in semiconductor manufacturing and engineering technologies have led to the inclusion of more than one CPU core in a single physical processor package. This, popularly know as Chip Multi Processing (CMP), allows multiple instruction streams to execute at the same time. CMP is in addition to today's Simultaneous Multi Threading (SMT) capabilities, like Intel® Hyper-Threading Technology which allows a processor to present itself as two logical processors, resulting in best use of execution resources. With CMP, today's Linux Kernel will deliver instantaneous performance improvement. In this paper, we will explore ideas for further improving peak performance and power savings by making the Linux Kernel Scheduler CMP aware.

1 Introduction

To meet the growing requirements of processor performance, processor architects are looking at new technologies and features focusing on enhanced performance at a lower power dissipation. One such technology is Simultaneous Multi-Threading (SMT). Hyper-Threading (HT) Technology[5] introduced in 2002, is Intel's implementation of SMT. HT delivers two logical processors running on the same execution core, sharing all the resources like functional execution units and cache hierarchy. This approach interleaves the execution of two instruction streams, making the most effective use of processor resources. It maximizes the performance vs. transistor count and power consumption.

Recent advances in semiconductor manufacturing and engineering technologies are leading to rapid increase in transistor count on a die. For example, forthcoming Itanium® family processor code named Montecito will have more than 1.7 billion transistors on a die! As the next logical step to SMT, these extra transistors are put to effective use by including more than one execution core with in a single physical processor package. This is popularly known as Chip Multi Processing (CMP). Depending on the number of execution cores in a package, it's either called a dual-core[4] (two execution cores) or multi-core (more than two execution cores) capable processors. In multi-threading and multi-tasking environment, CMP allows for significant improvement in performance at the system level.

In this paper, in Section 2 we will look at an overview of CMP and some implementation examples. Section 3 will talk about the generic OS scheduler optimization opportunities that are appropriate in CMP environment.

Linux Kernel Scheduler implementation details of these optimizations will be dwelled in Section 4. We will close the paper with a brief look at CMP trends in future generation processors.

2 Chip Multi Processing

In a Chip Multi Processing capable physical processor package, more than one execution core reside in a physical package. Each core has its own resources (architectural state, registers, execution units, up-to a certain level of cache, etc.). Shared resources between the cores in a physical package vary depending on the implementation. Some of the implementation examples are

a) each core could have a portion of on-die cache (for example L1) exclusively for itself and then have a portion of on-die cache (for example L2 and above) that is shared between the cores. An example of this is the upcoming first mobile dual-core processor from Intel, code named Yonah.

b) each core having its own on-die cache hierarchy and its own communication path to the Front Side Bus (FSB). An example of this is the Intel® Pentium® D processor.

Figure 1 shows a simplified block diagram of a physical package which is CMP capable, where two execution cores reside in one physical package, sharing the L2 cache and front side bus resources.

A physical package can be both CMP and SMT capable. In that case, each core in the physical package can in turn contain more than one logical thread. For example, a dual-core with HT will enable a single physical package to appear as four logical processors, capable of running four processes or threads simultaneously. Figure 2 shows an example of a CMP with two

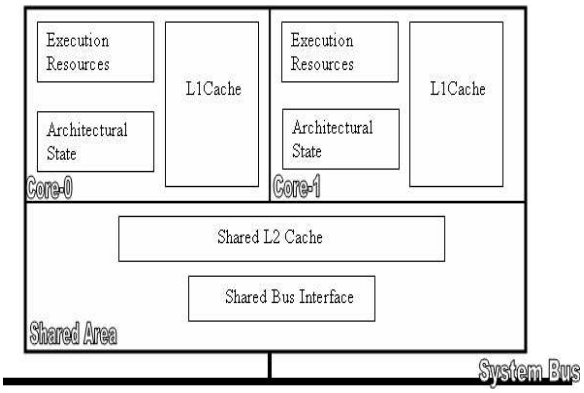

Figure 1: CMP implementation with two cores sharing L2 cache and Bus interface

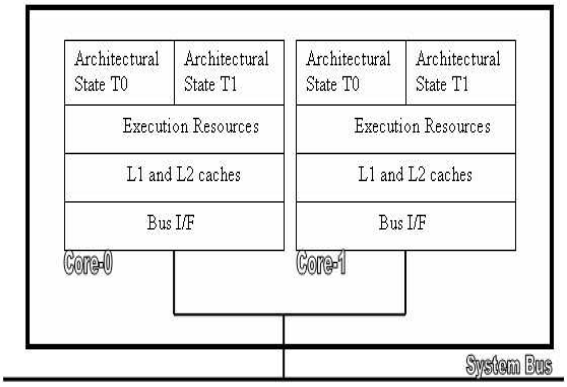

Figure 2: CMP implementation with two cores, each having two logical threads. Each core has their own cache hierarchy and communication path to FSB.

logical threads in each core and with each core having their own cache hierarchy and their own communication path to the FSB. An example of this is the Intel® Pentium® D Extreme Edition processor.

3 CMP Optimization opportunities

A multi-threaded application that scales well and is optimized on SMP systems will have an instantaneous performance benefit from CMP because of these extra logical processors coming from cores and threads. Even if the appli-

cation is not multi-threaded, it can still take advantage of these extra logical processors in a multi-tasking environment.

CMP also brings in new optimization opportunities which will further improve the system performance. One of the optimization opportunity is in the area of Operating System (OS) scheduler. Making the OS scheduler CMP aware will result in an improved peak performance and power savings.

In general, OS scheduler will try to equally distribute the load among all the available processors. In a CMP environment, OS scheduler can be further optimized by looking at micro architectural information(like L2 cache misses, Cycles Per Instruction (CPI), ...) of the running tasks. OS scheduler can decide which tasks can be scheduled on same core/package and which can't be scheduled together based on this micro architectural information. Based on these decisions, scheduler tries to decrease the resource contentions in a CPU core or a package and thereby resulting in increased throughput. In the past, some work[10, 9] has been done in this area and because of the complexities involved (like what micro architectural information need to be tracked for each task and issues in incorporating this processor architecture specific information into generic OS scheduler) this work is not quite ready for the inclusion in today's Operating Systems.

We will not address the micro architectural information based scheduler optimizations in this paper. Instead this paper talks about the OS CMP scheduler optimization opportunities in the case where the system is lightly loaded (i.e., the number of runnable tasks in the system are less compared to the number of available processors in the system). These optimization opportunities are simple and straight forward to leverage in today's Operating Systems and will help in improving peak performance or power savings.

3.1 Opportunities for improving peak performance

In a CMP implementation where there are no shared resources between cores sharing a physical package, cores are very similar to individual CPU packages found in a multi-processor environment. OS scheduler which is optimized for SMT and SMP will be sufficient for delivering peak performance in this case.

However, in most of the CMP implementations, to make best use of the resources cores in a physical package will share some of the resources (like some portion of cache hierarchy, FSB resources, ...). In this case, kernel scheduler should schedule tasks in such a way that it minimizes the resource contention, maximizes the system throughput and acts fair between equal priority tasks.

Let's consider a system with four physical CPU packages. Assume that each CPU package has two cores sharing the last level cache and FSB queue. Let's further assume that there are four runnable tasks, with two tasks scheduled on package 0, one each on package 1, 2 and package 3 being idle. Tasks scheduled on package 0 will contend for last level cache shared between cores, resulting in lower throughput. If all the tasks are FSB intensive (like for example Streams benchmark), because of the shared FSB resources between cores, FSB bandwidth for each of the two tasks in package 0 will be half of what individual tasks get on package 1 and 2. This scheduling decision isn't quite right both from throughput and fairness perspective. The best possible scheduling decision will be to schedule the four available tasks on the four different packages. This will result in each task having independent, full access to last level shared cache in the package and each will get fair share of the FSB bandwidth.

On CMP with shared resources between cores

in a physical package, for peak performance scheduler must distribute the load equally among all the packages. This is similar to SMT scheduler optimizations in todays operating systems.

3.2 Opportunities for improving power savings

Power management is a key feature in today's processors across all market segments. Different power saving mechanisms like P-states and C-States are being employed to save more power. The configuration and control information of these power saving mechanisms are exported through Advanced Configuration and Power Interface (ACPI)[2]. Operating System directed Configuration and Power Management (OSPM) uses these controls to achieve desired balance between performance and power.

ACPI defines the power state of processors and are designated as C0, C1, C2, C3, . . . , Cn. The C0 power state is an active power state where the CPU executes instructions. The C1 through Cn power states are processor sleeping (idle) states where the processor consumes less power and dissipates less heat.

While in the C0 state, ACPI allows the performance of the processor to be altered through performance state (P-state) transitions. Each P-state will be associated with a typical power dissipation value which depends on the operating voltage and frequency of that P-state. Using this, a CPU can consume different amounts of power while providing varying performance at C0 (running) state. At a given P-state, CPU can transit to numerically higher numbered C-states in idle conditions. In general, numerically higher the P states (i.e., lower the CPU voltage) and C-states, the lesser will be power consumed, heat dissipated.

3.2.1 CMP implications on P and C-states

P-states

In a CMP configuration, typically all cores in one physical package will share the same voltage plane. Because of this, a CPU package will transition to a higher P-state, only when all cores in the package can make this transition. P-state coordination between cores can be either implemented by hardware or software. With this mechanism, P-state transition requests from cores in a package will be coordinated, causing the package to transition to target state only when the transition is guaranteed to not lead to incorrect or non-optimal performance state. If one core is busy running a task, this coordination will ensure that other idle cores in that package can't enter lower power P-states, resulting in the complete package at the highest power P-state for optimal performance. In general, this coordination will ensure that a processor package frequency will be the numerically lowest P-state (highest voltage and frequency) among all the logical processors in the processor package.

C-states

In a CMP configuration with shared resources between the cores, processor package can be broken up into different blocks, one block for each execution core and one common block representing the shared resources between all the cores (as shown in Figure 1). Depending on the implementation, each core block can independently enter some/all of the C-state's. The common block will always reside in the numerically lowest (highest power) C-state of all the cores. For example, if one core is in C1 and other core is in C0, shared block will reside in C0.

3.2.2 Scheduling policy for power savings

Let's consider a system having two physical packages, with each package having two cores sharing the last level cache and FSB resources. If there are two runnable tasks, as observed in the Section 3.1 peak performance will be achieved when these two tasks are scheduled on different packages. But, because of the P-state coordination, we are restricting idle cores in both the packages to run at higher power P-state. Similarly the shared block in both the packages will reside in higher power C0 state (because of one busy core) and depending on the implementation, idle cores in both the packages may not be able to enter the available lowest power C-state. This will result in non-optimal power savings.

Instead, if the scheduler picks the same package for both the tasks, other package with all cores being idle, will transition slowly into the lowest power P and C-state, resulting in more power savings. But as the cores share last level cache, scheduling both the tasks to the same package, will not lead to optimal behavior from performance perspective. Performance impact will depend on the behavior of the tasks and shared resources between the cores. In this particular example, if the tasks are not memory/cache intensive, performance impact will be very minimal. In general, more power can be saved with relatively smaller impact on performance by scheduling them on the same package.

On CMP with no shared resources between the cores in a physical package, scheduler should distribute the load among the cores in a package first, before looking for an idle package. As a result, more power will be saved with no impact on performance.

4 Linux Kernel Scheduler enhancements

Process scheduler in 2.6 Linux Kernel is based on hierarchical scheduler domains constructed dynamically depending on the processor topology in the system. Each domain contains a list of CPU groups having a common property. Load balancer runs at each domain level and scheduling decisions happen between the CPU groups at any given domain.

All the references to "Current Linux Kernel" in the coming sections, stands for version 2.6.12-rc5[6]. Current Linux Kernel domain scheduler is aware of three different domains representing SMT (called `cpu_domain`), SMP (called `phys_domain`) and NUMA (called `node_domain`). Current Linux kernel has core detection capabilities for x86, x86_64, ia64 architectures. This will place all CPU cores in a node into different sched groups in SMP scheduler domain, even though they reside in different physical packages. The first step naturally is to add a new scheduler domain representing CMP (called `core_domain`). This will help the kernel scheduler identify the cores sharing a given physical package. This will enable the implementation of scheduling policies highlighted in Section 3.

Figure 3 shows the scheduler domain hierarchy setup with current Linux Kernel on a system having two physical packages. Each package has two cores and each core having two logical threads. Figure 4 shows the scheduler domain hierarchy setup with the new CMP scheduler domain.

4.1 Scheduler enhancements for improving peak performance

As noted in Section 3.1, when the CPU cores in a physical package share resources, peak per-

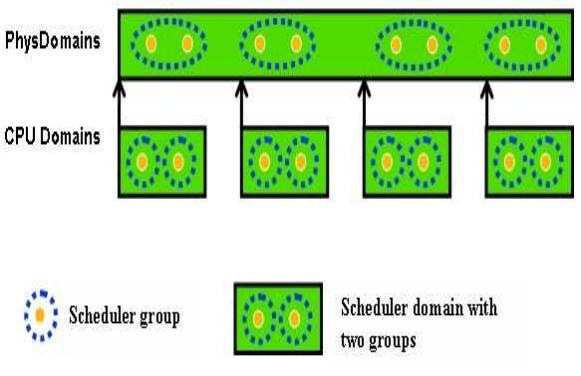

Figure 3: Scheduler domain hierarchy with current Linux Kernel on a system having two physical packages, each having two cores and each core having two logical threads.

formance will be achieved when the load is distributed uniformly among all physical packages. Following subsections will look into the enhancements required for implementing this policy.

4.1.1 Active load balance in presence of CMP and SMT

With SMT and SMP domains in current Linux Kernel, load balance at SMP domain will help in detecting a situation where all the SMT siblings in one physical package are completely idle and more than one SMT sibling is busy in another physical package. Load balance on processors in idle package will detect this situation and will kick active load balance on one of the non idle SMT siblings in the busiest package. Active load balance then looks for a package with all the SMT threads being idle and pushes the task (which was just running before active load balance got kicked in) to one of the siblings of the selected idle package, resulting in optimal performance.

Similarly in the presence of new scheduler domain for CMP, load balance in SMP domain will help detect a situation where more than one core in a package is busy, with another package being completely idle. Similar to the above, active load balance will get kicked on one of the non-idle cores in the busiest package. In the presence of SMT and CMP, active load balance needs to pick up an idle package if one is available; otherwise it needs to pick up an idle core. This will result in load being uniformly distributed among all the packages in a SMP domain and all the cores with in a package.

In pre 2.6.12 -mm kernels, there is a change in active load balance code which leverage the domain scheduler topology more effectively. Instead of looking for an idle package, active load balance code is modified in such a way, that it simply moves the load to the processor which detects the imbalance. In some of the cases[1] this will take few extra hops in finding a correct processor destination for a process but because of simplicity reasons this was pursued. This modification to active load balance also works in the presence of both SMT and CMP.

Figures 4 and 5 show how active balance plays a role in distributing the load equally among the physical packages and CPU cores in presence of CMP and SMT. Figure 6 shows how the new active balance will help in distributing the load equally among the physical packages, even though there is no idle package available. This will help from the fairness perspective.

4.1.2 cpu_power selection

One of the key parameters of a scheduler domain is the scheduler group's `cpu_power`. It represents effective CPU horsepower of the scheduler group and it depends on the underneath domain characteristics. With SMP and SMT domains in current Linux Kernel, `cpu_power` of sched groups in the SMP domain is

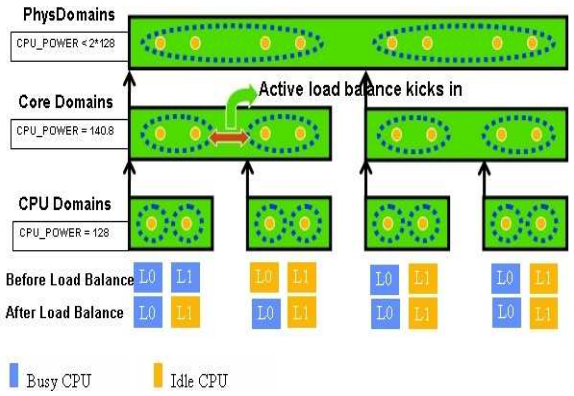

Figure 4: Demonstration of active load balance with 4 tasks, on a system having two physical packages, each having two cores and each core having two logical threads. Active load balance kicks in at the core domain for the first package, distributing the load equally among the cores

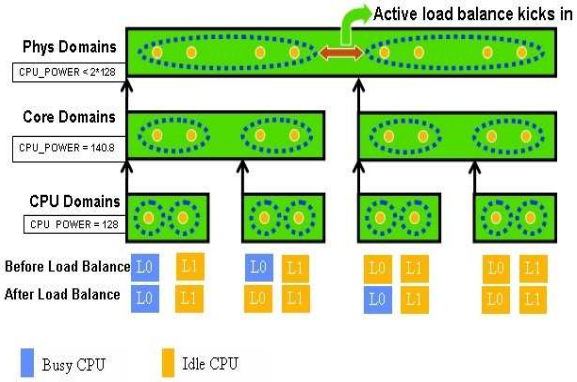

Figure 5: Demonstration of active load balance with 2 tasks, on a system having two physical packages, each having two cores and each core having two logical threads. Active load balance kicks in at SMP domain between the two physical packages, distributing the load equally among the physical packages

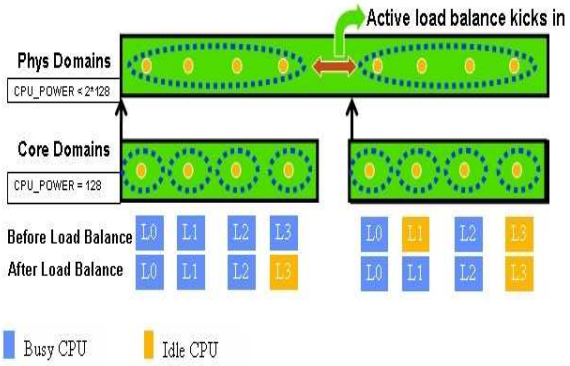

Figure 6: Demonstration of active load balance with 6 tasks, on a system having two physical packages, each having four cores. Active load balance kicks in at SMP domain between the two physical packages, distributing the load equally among the physical packages

calculated with the assumption that each extra logical processor in the physical package will contribute 10% to the `cpu_power` of the physical package.

With the new CMP domain, `cpu_power` for CMP domains scheduler group will be same as `cpu_power` of schedule group in current Linux Kernel's SMP domain (as the underneath SMT domain will remain same). Because of the new CMP domain underneath, new `cpu_power` for SMP domains sched group needs to be selected.

If the cores in a physical package don't share resources, then the `cpu_power` of groups in SMP domain, will simply be the horsepower sum of all the cores in that physical package. On the other hand, if the cores in a physical package share resources, then the `cpu_power` of groups in SMP domain has to be smaller than the no resource sharing case. We will discuss more about this in the power saving Sections 4.2.1 and 4.2.2 and determine how much smaller this needs to be for the peak performance mode policy.

4.1.3 exec, fork balance

Pre 2.6.12 mm kernels has exec, fork balance[3] introduced by Nick Piggin. Setting `SD_BALANCE_{EXEC, FORK}` flags to domains SMP and above, will enable exec, fork balance. Because of this, whenever a new process gets created, it will start on the idlest package and idlest core with in that package. This will remove the dependency on the active load balance to select the correct physical package, CPU core for a new task. This makes the process of picking the right processor more optimal as it happens at the time of task creation, instead of happening after a task starts running on a wrong CPU.

exec, fork balance will select the optimal CPU at the beginning itself and if dynamics change later during the process run, active load balance will kick in and distribute the load equally among the physical packages and the CPU cores with in them.

4.2 Scheduler enhancements for improving power savings

As observed in Section 3.2, when the system is lightly loaded, optimal power savings can be achieved when all the cores in a physical package are completely loaded before distributing the load to another idle package.

When the cores in a physical package share resources, this scheduling policy will slightly impact the peak performance. Performance impact will depend on the application behavior, shared resources between cores and the number of cores in a physical package. When the cores don't share resources, this scheduling policy will result in an improved power savings with no impact on peak performance.

For the CMP implementations which don't share resources between cores, we can make this power savings policy as default. For the other CMP implementations, we can allow the administrator to choose a scheduling policy offering either peak performance (covered in Section 4.1) or improved power savings. Depending on the requirements one can select either of these policies.

Following subsections highlights the changes required in kernel scheduler for implementing improved power savings policy on CMP.

4.2.1 cpu_power selection

The first step in implementing this power savings policy is to allow the system under light load conditions to go into the state with one physical package having more than one core busy and with another physical package being completely idle. Using scheduler group's `cpu_power` in SMP domain and with modifications to load balance, we can achieve this.

In the presence of CMP domain, we will set `cpu_power` of scheduling group in SMP domain to the sum of all the cores horsepower in that physical package. And if the load balance is modified such that, the maximum load in a physical package can grow up to the `cpu_power` of that scheduling group, then the system can enter a state, where one physical package has all its cores busy and another physical package in the system being completely idle.

We will leave the `cpu_power` for the CMP domain as before (same as the one used for SMP domain in the current Linux Kernel) and this will result in active load balance when it sees a situation where more than one SMT thread in a core is busy, with another core being completely idle. As the performance contribution by SMT is not as large as CMP, this behavior will be retained in power saving mode as well.

4.2.2 Active load balance

Next step in implementing this power savings policy is to detect the situation where there are multiple packages being busy, each having lot of idle cores and move the complete load into minimal number of packages for optimal power savings (this minimal number depends on the number of tasks running and number of cores in each physical package).

Let's take an example where there are two packages in the system, each having two cores. There can be a situation where there are two runnable tasks in the system and each end up running on a core in two different packages, with one core in each package being idle. This situation needs to be detected and the complete load needs to be moved into one physical package, for more power savings.

For detecting this situation, scheduler will calculate watt wastage for each scheduling group in SMP domain. Watt wastage represents number of idle cores in a non-idle physical package. This is an indirect indication of wasted power by idle cores in each physical package so that non-idle cores in that package run unaffected. Watt wastage will be zero when all the cores in a package are completely idle or completely busy. Scheduler can try to minimize watt wastage at SMP domain, by moving the running tasks between the groups. During the load balance at SMP domain level, if the normal load balance doesn't detect any imbalance, idle core (in a package which is not wasting much power compared to others in SMP domain) can run this power saving scheduling policy and see if it can pull a task (using active load balance) from a package which is wasting lot of power.

In the last example, idle core in package 0 can detect this situation and can pickup the load from busiest core in package 1. To prevent the idle core in package 1 doing the same thing to the busiest core in package 0 (causing unnecessary ping-pong) load balance algorithm needs to follow the ordering. Figure 7 shows a demonstration of this active load balance, which will result in improved power savings.

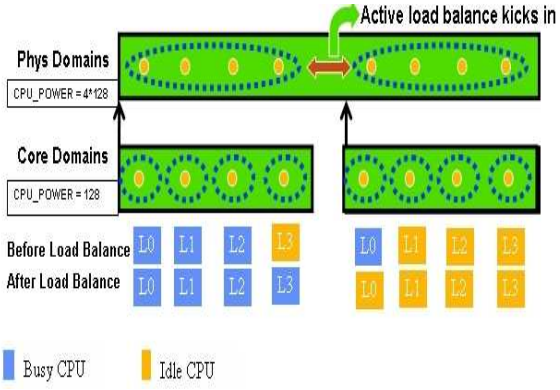

Figure 7: Demonstration of active load balance for improved power savings with 4 tasks, on a system having two physical packages, each having four cores. Active load balance kicks in between the two physical packages, resulting in movement of the complete load to one physical package, resulting in improved power savings

As the number of cores residing in a physical package increase, shared resources between the cores will become bottle neck. As the contention for the resources increase, power saving scheduling policy will result in an increased impact on peak performance. As shown in Figure 7, moving the complete load to one physical package will indeed consume lesser power compared to keeping both the packages busy. But if the cores residing in a package share last level cache, the impact of sharing the last level cache by 4 tasks may outweigh the power saving. To limit such performance impact, we can let the administrator choose the allowed watt wastage for each package. Allowed watt wastage is an indirect indication of the scheduling group's horsepower. `cpu_power` of the scheduling group in SMP domain can be mod-

ified proportionately based on the allowed watt wastage. Load balance modifications in Section 4.2.1 will limit the maximum load that a package can pickup (under light load conditions) and hence the impact to peak performance. More power will be saved with smaller allowed watt wastage. In the case shown in Figure 7, administrator for example can say, under light load conditions don't overload one physical package with more than 2 tasks.

Setting the scheduler groups `cpu_power` of SMP domain to the sum of all the cores horsepower (i.e., allowed watt wastage is zero) will result in a package picking up the max load depending on the number of cores. This will result in maximum power saving. Setting the `cpu_power` to a value less than the combined horsepower of two cores (i.e., allowed watt wastage is one less than the number of cores in a physical package) will distribute the load equally among the physical packages. This will result in peak performance. Any value for `cpu_power` in between will limit the impact to peak performance and hence the power savings.

Administrator can select the peak performance or the power savings policy by setting appropriate value to the scheduler group's `cpu_power` in SMP domain.

4.2.3 exec, fork balance

`SD_BALANCE_{EXEC, FORK}` flags need to be reset for domains SMP and above, causing the new process to be started in the same physical package. Normal load balance will kick in when the load of a package is more than the package's horsepower (`cpu_power`) and there is an imbalance with respect to another physical package.

5 Summary & Future work

CMP related scheduler enhancements discussed in this paper fits naturally to the 2.6 Linux Kernel Domain Scheduler environment. Depending on the requirements, administrator can select the peak performance or power saving scheduler policy. We have prototyped peak performance policy discussed in this paper. We are currently experimenting with the power saving policy, so that it behaves as expected under the presence of CMP, SMT and under the light, heavy load conditions. Once we complete the performance tuning and analysis with real world workloads, these patches will hit the Linux Kernel Mailing List.

For the future generation CMP implementations, researchers and scientists are experimenting[8] with "many tens of cores, potentially even hundreds of cores per package and these cores supporting tens, hundreds, maybe even thousands of simultaneous execution threads." Probably we can extend Moore's law[7] to CMP and can dare say that number of cores per die will double approximately every two years. This sounds plausible for the coming decade at least. With more CPU cores per physical package, kernel scheduler optimizations addressed in this paper will become critical. In future, more experiments and work need to be focused on bringing micro architectural information based scheduling to the mainline.

Acknowledgments

Many thanks to the colleague's at Intel Open Source Technology Center for their continuous support.

Thanks to Nick Piggin and Ingo Molnar for always providing quick comments on our scheduler patches.

References

[1] Active load balance modification in pre 2.6.12 -mm kernels. http://www.ussg.iu.edu/hypermail/linux/kernel/0503.1/0057.html.

[2] Advanced configuration and power interface spec 3.0. http://www.acpi.info/DOWNLOADS/ACPIspec30.pdf.

[3] Balance on exec and fork in pre 2.6.12 -mm kernels. http://www.ussg.iu.edu/hypermail/linux/kernel/0502.3/0037.html.

[4] Intel dual-core processors. http://www.intel.com/technology/computing/dual-core.

[5] Intel hyper-threading technology. http://www.intel.com/technology/hyperthread.

[6] Linux kernel. http://www.kernel.org.

[7] Moore's law. http://www.intel.com/research/silicon/mooreslaw.htm.

[8] Processor and platform evolution for the next decade. http://www.intel.com/technology/techresearch/idf/platform-2015-keynote.htm.

[9] Daniel Nussbaum Alexandra Fedorova, Christopher Small and Margo Seltzer. *Chip Multithreading Systems Need a New Operating System Scheduler*. SIGOPS, ACM, 2004.

[10] Jun Nakajima and Venkatesh Pallipadi. *Enhancements for Hyper-Threading Technology in the operating System: Seeking the Optimal Scheduling*. WIESS, USENIX, December 2002.