Multi-core and Linux* Kernel

Suresh Siddha
Intel Open Source Technology Center

Abstract

Semiconductor technological advances in the recent years have led to the inclusion of multiple CPU execution cores in a single processor package. This processor architecture is known as Multi-core (MC) or Chip Multi Processing (CMP).

Any application which is well optimized and scales with SMP will take immediate benefit of the multiple execution cores, provided by the Multi-core architecture. Even if the application is single-threaded, multi-tasking environment will take advantage of these multiple execution cores.

2.6 Linux kernels (which have better SMP scalability compared to 2.4 kernels) take instant advantage of the MC architecture. MC also brings in performance optimization opportunities which will further enhance the performance.

This paper captures the recent enhancements to the 2.6 Linux Kernel which better support and enhance the performance of Multi-core capable platforms.

1. Introduction

In Multi-core processor based platforms, more than one execution core reside in a physical package. Each core has its own resources (architectural state, registers, execution units, some or all levels of caches, etc.). Shared resources between the cores in a physical package vary depending on the implementation. Typical MC implementations share the last level cache and the front side bus (FSB) resources.

Duplicate execution resources in MC capable processors, allow parallel execution of the application threads and hence offer significantly greater concurrency. MC capable processors also offer greater system density, and performance per watt compared to the single core processor packages that they replace. Consequently, they are poised to bring new levels of performance and scalability. Most of the programming challenges for multi-core environment emanate from SMP environment. Enhancements that have gone into 2.6 Linux Kernel for improving SMP scalability greatly help in MC environment too. However the presence of shared resources (e.g. last level cache, front side bus resources, power management states, etc between CPU cores residing in a physical package) in MC environment, brings in additional challenges. Addressing these challenges will lead to lower shared resource contention and improvement in the peak performance.

Section 2 will look into the Multi-core topology identification mechanism in 2.6 Linux Kernel. Section 3 will look into the 2.6 Linux Kernel scheduler and talks about the Multi-core related scheduler enhancements. Finally the paper will close with a summary.

2. Detecting Hardware Multi-threading and Multi-core topology in 2.6 Linux

Multiple logical threads (HT) or execution cores (MC) in a physical package appear to the operating system as multiple logical processors (similar to SMP environment). For example, Dual-Core Intel Xeon® processor 7100 series provides four logical processors for each physical package in the platform, as it is a dual-core processor with HT Technology enabled.

Linux kernel exports the multi-threading and multi-core topology to the applications using two mechanisms. Applications can use the topology information for number of purposes such as application licensing, binding application threads/processes to a specific group of threads/cores/physical packages to achieve peak throughput.

2.1 Exporting through 'proc' file system:

/proc/cpuinfo contains information about the different CPUs in the system. This proc file also exports the multi-threading and multi-core topology information as seen by the OS. This is the legacy mechanism of exporting topology information to the user, which started initially when Hyper-threading got introduced.

"ht" in 'flags' field of /proc/cpuinfo indicate that the processor supports the Machine Specific Registers to report back HT or multi-core capability. Additional fields (listed down below) in the CPU records of /proc/cpuinfo will give more precise information about the CPU topology as seen by the operating system.

"physical id"	Physical package id of the logical CPU
"siblings"	Total number of logical processors(includes both threads and cores) in the physical package currently in use by the OS
"cpu cores"	Total number of cores in the physical package currently in use by the OS
"core id"	Core id of the logical CPU

Legacy HT optimized applications parse "ht" flag and "physical id", "siblings" fields in /proc/cpuinfo for identifying physical packages in the system. This package identification will be used for example in application licensing purposes or in binding a process to a specific physical package. These applications require no change and will work as it is on a multi-threaded and/or multi-core system if they require only the logical CPU to a physical package mapping.

To identify if the physical package is purely multi-threaded or purely multi-core capable or both, then in addition to the "ht" flag, applications need to parse all the above mentioned fields. Following logic can be used to identify the CPU capabilities as seen by the OS.

"siblings == cpu cores >= 2"
- Indicates that the physical package is multi-core capable and is not Hyper-threading capable

"siblings >= 2" && "cpu cores == 1"
- Indicates that the physical package is Hyper-threading capable and has one cpu core

"siblings > cpu cores > 1"
- Indicates that the physical package is both Hyper-threading and multi-core capable

To build the complete topology of which logical cpus belong to which CPU core and/or physical package, application need to parse "physical id" and "core id" fields. For two logical cpus, if both these ids are same, then they belong to same core (Hyper-Threading siblings). If they have same physical package id and different core id, then they belong to same physical package (core siblings).

2.2 Exporting through 'sysfs' file system

More recent Linux Kernels [LK] (like 2.6.17) have the CPU topology exported in sysfs as well. This mechanism is simpler and faster compared to the previously mentioned /proc interface. Below listed fields exported under /sys/devices/system/cpu/cpuX/topology/ provide the complete topology information.

physical_package_id	Physical package id of the logical CPU
core_id	Core id of the logical CPU
core_siblings	Siblings mask of all the logical CPUs in a physical package
thread_siblings	Siblings mask of all the logical CPUs in a CPU core

Hamming Weight (number of bits set) of siblings mask will give the physical package capabilities.

“HW(core_siblings) == HW(thread_siblings) >= 2”
- Indicates that the physical package is Hyper-threading capable and has one cpu core

“HW(core_siblings) >= 2” && “HW(thread_siblings) == 1”
- Indicates that the physical package is multi-core capable and is not Hyper-threading capable

“HW(core_siblings) > HW(thread_siblings) > 1”
- indicates that the physical package is both Hyper-threading and multi-core capable

3. 2.6 Linux Process Scheduler

The most significant change in 2.6 Linux Kernel which improved scalability in multi processor system was in the kernel process scheduler. The design of Linux 2.6 scheduler is based on per cpu runqueues and priority arrays, which allow the scheduler perform its tasks in O(1) time. This mechanism solved many scalability issues but the scheduler still didn’t perform as expected on Hyperthreaded systems and on higher end NUMA systems. In case of Hyper-threading, more than one logical CPU shares the processor resources, cache and memory hierarchy. And in case of NUMA, different nodes have different access latencies to the memory. These non uniform relationships between the CPUs in the system pose significant challenge to the scheduler. Scheduler must be aware of these differences and the load distribution needs to be done accordingly.

To address this, 2.6 Linux kernel scheduler introduced a concept called scheduling domains [SD]. 2.6 Linux kernel used hierarchical scheduler domains constructed dynamically depending on the CPU topology in the system. Each scheduler domain contains a list of scheduler groups having a common property. Load balancer runs at each domain level and scheduling decisions happen between the scheduling groups in that domain. On a high end NUMA system with processors capable of Hyper-threading, there will be three scheduling domains, one each for HT, SMP and NUMA.

In the presence of Hyperthreading, when the system has fewer tasks compared to number of logical CPUs in the system, scheduler must distribute the load uniformly between the physical packages. This distribution will avoid scenarios in the system where one physical package has more than one logical CPU busy and another physical package is completely idle. Uniform load distribution between physical packages will lead to lower resource contention and higher throughput. Presence of Hyperthreading scheduler domain will help the scheduler achieve the equal load distribution between the physical packages.

Similarly the NUMA scheduling domain will help in unnecessary task migration from one node to another. This will ensure that the tasks will stay most of the time in their home (where the task has allocated most of its memory) node.

3.1 MC aware Linux Kernel Scheduler:

2.6 Linux Kernel domain scheduler up to 2.6.16, is aware of three different domains representing HT, SMP and NUMA. On a MC system, Linux kernel which has multi-core detection capabilities will place cpu cores and physical packages in SMP scheduler domain. So is a new scheduler domain representing MC [OLS2005] required? Following sections will answer this question.

3.1.1 Opportunities for improving peak performance:

In a MC 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 doesn't have to do anything more in this case as far as performance is concerned.

However, in most of the MC implementations, to make best use of the resources and to make inter core communication more efficient, cores in a physical package will share some of the resources(for example Intel® Core™ Duo processor has two CPU cores sharing the L2 Intel® Smart Cache[CACHE]). In this case, the kernel scheduler should schedule tasks in such a way that it minimizes the resource contention, maximizes the resources utilization and maximizes the system throughput.

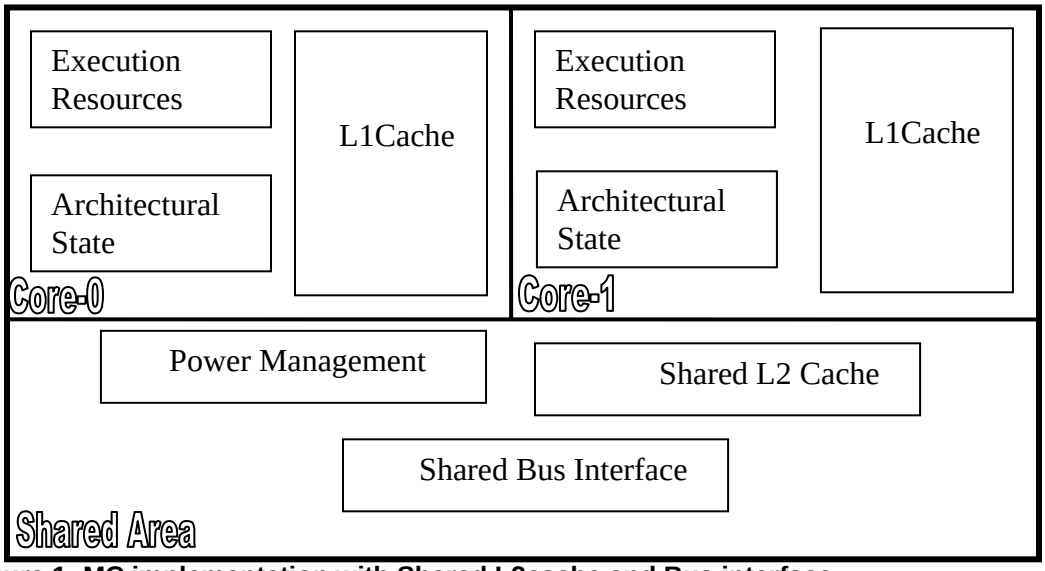

Figure 1: MC implementation with Shared L2cache and Bus interface.

Let's consider a hypothetical system with two physical packages in the system, with a common FSB. Let us assume that each package has two cores sharing the last level cache and FSB queue. Let us further assume that we have two runnable, totally independent, tasks to schedule and the scheduler schedules two tasks to package 0, leaving package 1 idle. In this scenario, tasks scheduled on package 0 will contend for last level cache shared between cores, resulting in lower throughput. Also resources in package 1 are left unutilized. This scheduling decision isn't quite right from the resource utilization perspective.

The best possible scheduling decision will be to schedule two tasks on two different packages. This will result in each task having independent, full access to last level shared cache in the package and fair share of the FSB Bandwidth leading to proper resource utilization.

So in the case where the cores in a package share resources and when the system is lightly loaded, scheduler needs to distribute the load equally among all the packages to achieve peak performance.

3.1.2 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. Advanced Configuration and Power Interface (ACPI) [ACPI] defines the power state of processors (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. While most people think of P states in terms of reducing the frequency of the processor when not fully busy, in reality it’s a more complex combination of reduced frequency and voltage. Using these P states, a CPU can consume different amounts of power while providing different performance at C0 (running) state. At a given P-state, cpu can transit to higher C-states in idle conditions. In general, higher the P and C-states, the lesser will be power consumed, heat dissipated.

MC implications on P and C-states:

P-states:

In a MC configuration, typically all cores in one physical package will share the same voltage; there is only one voltage regulator per socket present on the motherboard. Hence P-state transitions (which impact both frequency and voltage) for all the cores need to happen at the same time. This coordination of P-states 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 when the transition is guaranteed to not lead to incorrect or non-optimal performance state. If one core is 100% busy running a task, this coordination will ensure that other idle cores in that package can't enter low 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 typical MC configuration, 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. Since cores operate independently, each core block can independently enter a c-state. For example, one core can enter C1 or C2 while the other executes code in C0. 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 C2 and other core is in C0, shared block will reside in C0.

Scheduling policy for power savings:

Let's take the same hypothetical system which was considered before, having two physical packages with each package having two cores sharing the last level cache and FSB resources. If we have two runnable tasks, as observed in the previous section, peak performance will be achieved when these two tasks are scheduled on different packages. But, because of the P-state coordination, we are restricting other 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). This will result in non-optimal performance from power saving's perspective.

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, we can save more power with relatively smaller impact on performance by scheduling them on the same package.

3.2 Scheduler domain for MC:

For implementing the above mentioned performance and power savings policies, Linux kernel scheduler needs to differentiate between cpu cores and physical packages. A new scheduler domain representing MC will achieve this differentiation and will enable these scheduler policies.

MC Scheduler domain representing shared last level cache between CPU cores is added in 2.6.17 Linux kernel. Figure 2 demonstrates default peak performance scheduling policy which is present in 2.6.17 Linux kernel. 4 tasks are running on a system having two physical packages, each having two cores (sharing last level cache) and each core having two logical threads. Load balance kicks in at the MC domain for the first package, resulting in equal load distribution among the cores.

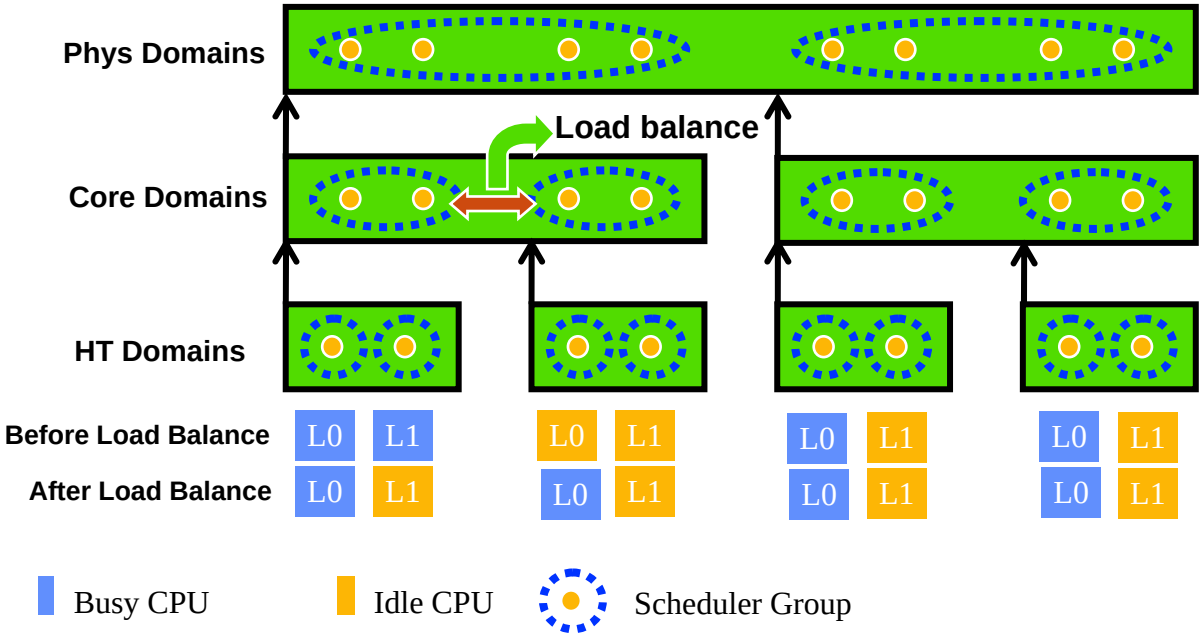

Figure: 2 Demonstration of MC Scheduler Peak Performance policy on a DP system with each physical package having two execution cores and each core having two logical threads.

MC power savings policy is added in Linux 2.6.18-rc1. sysfs entries 'sched_mc_power_savings' and 'sched_smt_power_savings' in /sys/devices/system/cpu/ control the multi-core/multi-threading power savings policy for the scheduler. When power savings policy is enabled and under light load conditions, scheduler will minimize the physical packages/cpu-cores carrying the load and thus conserving power (with a performance impact based on the workload characteristics)

Figure 3 demonstrates load balance for improved power savings with 4 tasks, on a system having two physical packages, each having four cores. 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 with a performance impact which will depend on the behavior of the tasks and shared resources between the cores.

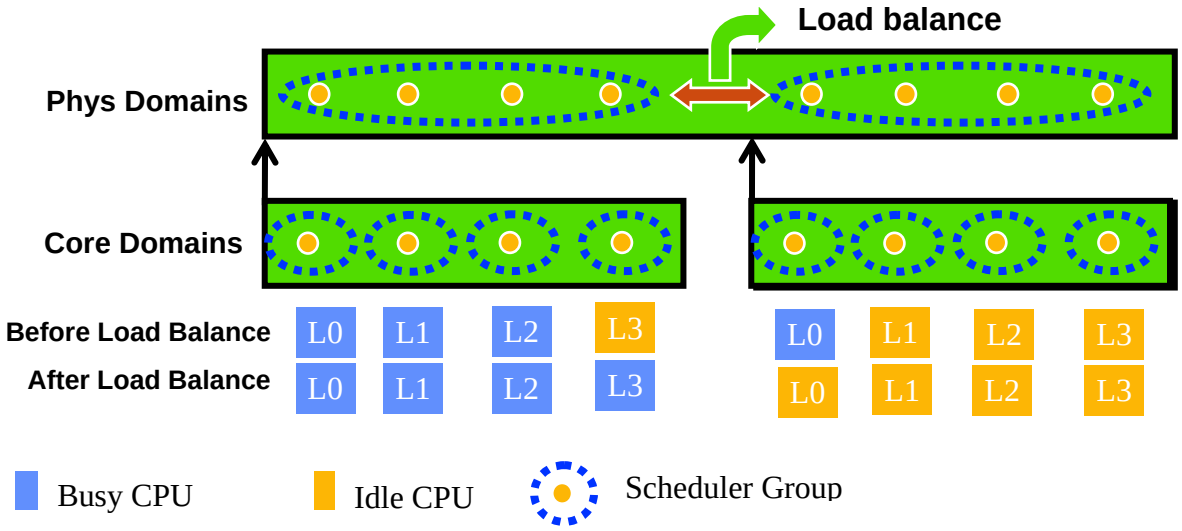

Figure 3: Demonstration of MC Scheduler Power savings policy on a DP system with each physical package having four execution cores.

4. Summary

Operating systems and Applications which scale well with large number of CPUs in SMP environment will take instantaneous benefit of Multi-core architecture. Applications can use the core and thread topology information exported by the OS for licensing and other purposes like task binding. With number of cores per physical package and the shared resources between them increase, kernel scheduler optimizations discussed in this article will become critical.

References

[ACPI] ACPI http://www.acpi.info/
[CACHE] Intel® Smart Cache http://www.intel.com/products/processor/coreduo/smartcache.htm
[LK] Linux Kernel http://www.kernel.org
[OLS2005] Chip Multi Processing (CMP) aware Linux Kernel Scheduler – OLS 2005 - Suresh Siddha, Venkatesh Pallipadi & Asit Mallick
[SD] Scheduling Domains http://lwn.net/Articles/80911/