

SMP, CMP & SMT and Operating systems

Authored By:

Peter Chacko,
Founder & CTO
Netdiox Computing Systems
[A division of Sciendix data systems Pvt.Ltd]
peter@nediox.com
www.netdiox.com

Table of Contents

1. Introduction-----	3
2. SMP,CMP,SMT advancements and Linux Scalability-----	3
3. Scheduler Domain-----	5
4. Multi processing and MESI protocol-----	6
5. Multi core and caching performance-----	7
6. SMPs and Distributed interrupt processing. -----	7
7. Multi-core and power saving-----	8
8. Virtualized SMPs and beyond.-----	8
9. About the Author-----	9

1.

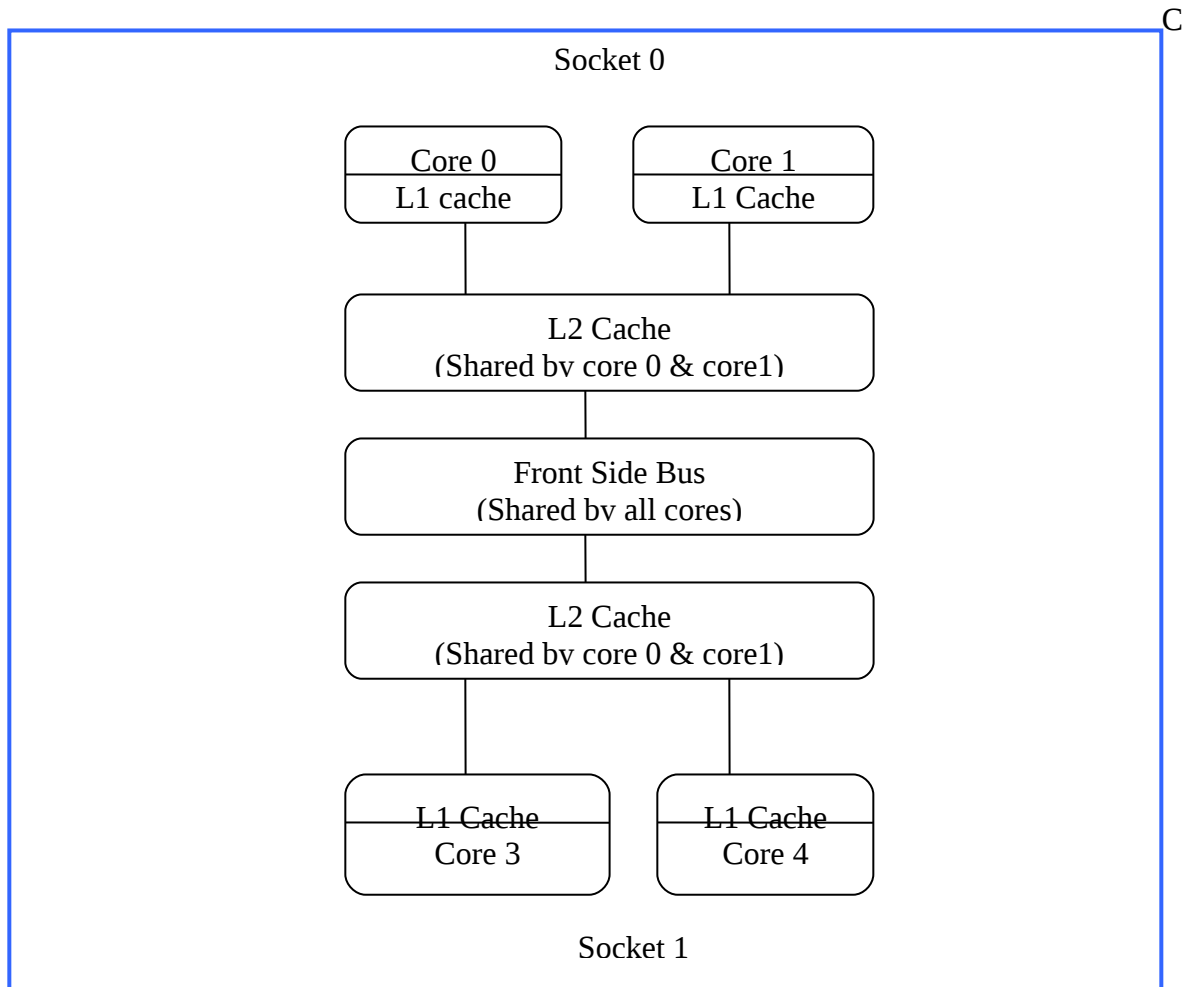
Introduction

As the advancement of hardware technologies now move towards the integration of more cores in a single die, to increase the effective performance of the processor to user application, study of multi-core systems is imperative. Moore's law has reached a T junction, thanks to power / thermal constraints of increasing per-cpu clock frequency and we are entering in to a new world of computing. Operating systems, applications compilers libraries hypervisors and related system/application software now has to accommodate this disruptive evolution of hardware technologies. This document talks about multi-core technologies in general. Linux process scheduler changes in a multi-core environment are briefly covered, and then the data coherence issues covered in detail. Paper is concluded with the author's views towards current and the future innovations around virtual SMP.

2. SMP CMP SMT –Introduction

We are familiar with the single core, sequential programming and have learned that UNIX is a multi-tasking operating system. And those people are considered as very technical who knew multi-tasking is just an illusion and that at any instant only one task can execute. But that has changed today. Parallelism/concurrency is now the fundamentals of computing. In a single mother board, we can now find multiple CPUs each has multiple cores where each core itself is capable of running multiple tasks in parallel with hardware multi-threading. The ability to fabricate multiple, homogeneous CPUs as one large computer is called Symmetric Multi Processing, the art of bundling multiple cores in single die is called Chip Level Multi Processing(CMP) and the third category of parallel execution of multiple threads in a single core is called Simultaneous Multi Threading(SMT). Usually SMT is exploited to realize instruction level parallelism(ILP) though it is best achieved at super scalar micro architecture level. With CMP, we only achieve thread level parallelism (where multiple threads are executed in parallel (TLP)).

Consider the following picture

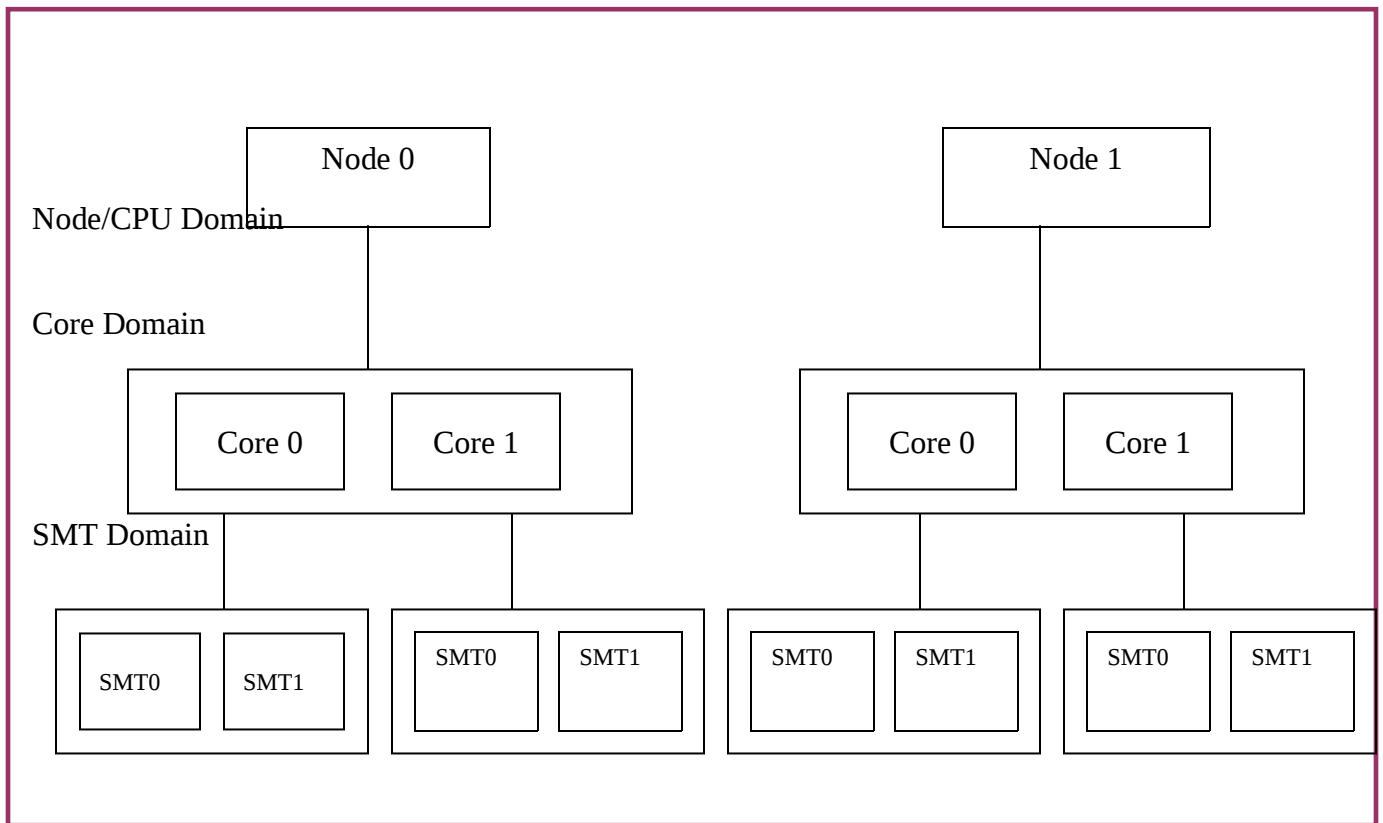


As shown above , a single user thread can now be run on any one of the 4 cores. If each core is SMT-enabled , it will be running on that logical processor of one of any cores. Each logical core has it's own hardware registers. ALU/co-processors sharing all layers of caches with other siblings thread of the same core. The entire package contains 2 CPUs. Every CPU has 1L2 cache which is shared by the cores in it. Each core has it's own L1 cache. All CPU's share the memory, and front side Bus. So now the challenge in developing highly efficient, multi-core aware software means, writing system software in such a way that contention to shared resources (FSB,L2 cache) are minimized, cache locality is considered when different threads are scheduled , OS assisted power management functionality is fully exploited when selectively shutting down some packages, and intelligent CPU schedulers are needed to avoid raises/locking issues .It is not easy and these changes will happen over a time on an evolutionary basis. In the next section , we will look at the scheduling enhancements made

by Linux kernel community to exploit SMP-CMP-CMT with the introduction of scheduler domains.

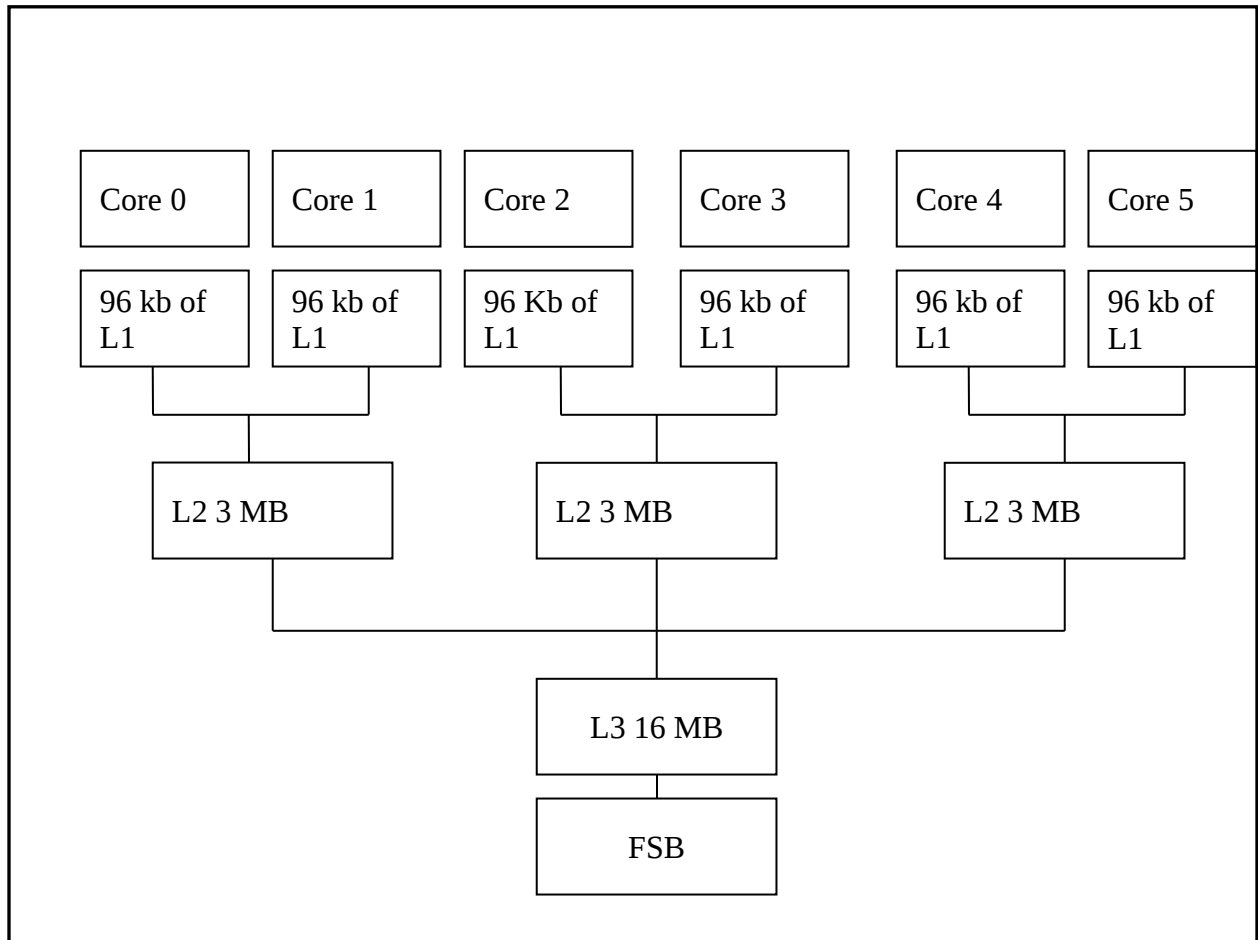
3. Scheduler domain

Before we proceed , Let's ask this question. What is the difference between the productivity of an existing engineer and new hire, who brings experience from another company ? You know the answer. Same thing applies here. Whenever a thread is elected to be scheduled , scheduler now has to choose the core it ran last time, to dispatch the thread so that already cached L1/L2 data is exploited to hide the memory access latency. Virtual memory manager module now has to make sure that if it allocates a page to a thread from a NUMA node , the thread in question is scheduled to run on the same processor, if possible. Scheduler now has to trade off performance versus power when it decides to move all threads to a single package and turn of the other CPU off. Idea of multi-core aware scheduling/ load balancing is realized through scheduler domains. Consider the following picture of a physical board with 2 NUMA nodes, having 2 cores each , having 2 SMT threads each.



As shown above tree structured logical view of virtual processors , In reality there is 2*4 virtual processors . In a perfect world , idea of scheduler domain is to prevent any thread from being scheduled out of its domain. For example , if we decide to run a thread from the sleep queue, we first look at its own domain when it ran last, and the logical processors (Lower most domain) of the same core are tried. If it is not possible, we look for other another core, and then another Node. Hence we always look for scheduling in the same core and then another core and then another CPU (node domain), to make use of Node/Core/SMT locality.

Following is the schematic diagram of Dunnington processor by intel..



Dunnington of Intel 6-core(Source:Intel Technical Papers)

Essentially, sched-domain and sched-group structures of the new Linux scheduler provide enough abstraction to implement policies that govern intelligent load balancing / scheduling of different tasks across processors of varying cost metrics(each warmth, NUMA heterogeneity etc.). Please take a look at kernel source code to see the detailed software implementation of the technique, if you are a kernel developer (kernel/scheduler.c, host all the code for implementing this. (Besides take a look at the paper, “Linux on NUMA systems, written by Martin et al).

4. Multi processing and MESI protocol

MESI protocol has been developed to implement cache consistency algorithm at hardware level. Different caches of the same CPUs/different CPUs need to be consistent when it contains common data while multiple CPUs are operating.

MESI is the acronym for Modified .Exclusive, Shared and Invalid which indicates the four states of the protocol. Processor can modify these status (due to a READ/WRITE operation. Or through an internal snooping or through an external elements(other processors or L2 cache controller through external snooping)..The data of cache line will be marked *modified*

if it is a single cache line ,in the entire system. The cache line can be read or written to, without going through an external BUS cycle. Contents of RAM and the cache line may not be same. Exclusive cache line is similar, except the fact that contents are same as that of in the RAM. When a write access issues, it becomes a Modified cache line. *Shared* cache line is shared across all CPUs . When CPU writes into the cache line, it is switched through an external BUS cycle modifying the RAM, as well as invalidating other caches. (other cache lines get the external snooping from this cache controller and gets it cache lines that contains the same data gets invalidated).*Invalid* This cache line contains stale data Read access always cause a cache miss ,while WRITE access will be switched through an external BUS cycle. MESI doesn't support a WRITE-ALLOCATION caching strategy.

5. Multi core and caching performance

Caching across multi-core is double edged sword. On one side it increases the execution speed as data misses are reduced. On the other hand when multiple cores modify shared caches cache lines are ping-ponged as well as memory BUS is contented. Intel Architecture offers prefetch family of instruction to enable software instruct cache controller to bring the corresponding cache line refilled. When it is used without care, performance can be reduced. Sometimes the new cache line may replace an existing cache line which the application may need before it needs the memory region for which it issued prefetch. Always make sure that related frequently accessed variables are stored in a single cache line. To make the best use of multi-core caching , it needs a totally redesigned multi-threaded software that align well with multi-core way. It is always important that you groups related functions together, so that the callee and the caller can be served from the cache lines which can be re-filled and prefetched together. If it backs memory addresses which are close together.. (I-cache). When you design functions in small sizes it is likely that it is contained few lines which are re-filled in parallel with hardware predication. Compiler optimizers should align hot sections of the code in a single line. Intel offers various pragma directives (on Intel compiler) to an an automatic convert code to be multi-core efficient. Multi-core is MIMD(Multiple Instructions, Multiple Data). Change the way you look at programming. If you write the sequential code, and cannot expect a large scale CMP execute it efficiently. One needs to think of a big picture in terms of how various sections of the code, data operates on each other across various threads. When you bring synchronization primitives with different locks always make sure that what lock acquisition/release happens without much cache-line bouncing. Data Partitioning, lock-free data structures, cooperative scheduling of various threads all have a big role to play here. It is an evolutionary industry and lots of opportunities come with it. Different solutions may work differently for different workloads and Data access pattern. So lots of invocations at all levels can be expected.

6. SMPs and Distributed interrupt processing.

Intel and other vendors have developed multi-APIC(Advanced , programmable interrupt controllers) newer chips to stand in for 8259A style PICs for SMP systems, having each CPU has its own APIC that serves the interrupt delivered to it by the master APIC, through ICC bus. Essentially IO APIC now plays the role of an interrupt router. The IO APIC has a data structure (interrupt redirection table having several(typically 24) entries)each entry can be programmed to hold the interrupt vector and priority, destination processor and the processor

selection mode. This information is used to convert an interrupt signal from the external device to a message delivered to local APIC. IRQs can either be delivered statically to different CPUs based on the Redirection table entries or based on the priority of a process a specific processor is executing. (The CPU in question has to program task priority register, each time it does a context switch, to publish this information to the IO APIC). Inter-processor interrupts are delivered from CPU to CPU, by first programming ICR (interrupt command register) of its own APIC, which causes a message to be delivered through the ICC bus to the target CPU (ICR will tell who the target is). Target CPU's APIC just receives the message and delivers the needed interrupt to its own master (target CPU). (Mechanisms like TLB shoot-downs, remote pre-emption of processes are implemented using IPI exchanges).

7. Multi-core and power saving

When no. of threads are less than no. of cores (which are usually the case when the no. of cores exceeds 16 or so and during off-peak hours), we can do another optimization, to save the power by scheduling all the threads to all the cores in a single package, and bringing to another package to C0/P0 state. If the system performance expectations allow, we can even think of overloading a selected package and bringing down other packages, trading off performance for power. This can be done for the nightly batch jobs (like software builds/automated test execution etc) which are not mission-critical. The ACPI module enables these power-saving mechanisms. OSPM (Operating System directed Power Management) works on top of ACPI to achieve the needed balance between performance and power. Power states are designated as C0, C1, C2... etc Cn. C0 is being active (when CPU can execute instructions) and C1...Cn are various states of stepping saving power and reducing heat. While between the C0 state, CPU can be in various P-states using varying amounts of power. Another way an intelligent scheduler can schedule various threads based on the micro-architectural state like no-cache-misses, TLB flushes, MESI invalidations etc. Various processor manufacturers are adding more registers to implement this fine-granular scheduling and we can expect more technologies in this direction in the future...and related bugs where we notice that suddenly some of the cores are powered down, when many threads are waiting to be scheduled. ☺

8. Virtualized CMPs/SMPs and beyond.

Things turn very messy when it comes to virtualized CMPs/SMPs. With hypervisors, now CPU schedulers have a bigger role to play in allocating multiple cores to multiple VMs... In a virtualized world, each physical CPU is typically 1-to-1 mapped for SMP VMs. Many virtualization solutions only support single-CPU VM (where all threads of the VM are scheduled to run only on a single physical CPU). When an SMP VM selects a physical processor as the backing processor by the VMM scheduler, we say the VM is now scheduling a virtual CPU or VCPU. Should the VMM scheduler give all VCPUs to different threads of the same VM or different threads of different VMs? What happens if a thread is scheduled on a VCPU but it needs more threads from the same VM for its operation? Today, no VMM has a perfect solution. Leading system VM vendors typically employ a co-scheduler

(where all threads of the same VM are run at the same time or co-scheduled in different VCPUs mapped to different PCPUs. Here we need a multi-core, aware, Virtualization aware, process concurrency relations ships aware, cache-warmth aware, process readiness-aware scheduler....Tough problem to solve...opportunity for many inventors....Its better to have the hypervisor does all CPU allocation. Then what is the role of a scheduler within a VM ? How will it co-exist with another guest VM that has a different policy in selecting the core ?

Why do we need an OS scheduler in a hypervisor environment ? CPU architects, Virtualization experts, OS experts have to meet and really come up with an inter-disciplinary ways and lets call it next generation SMP schedulers.. And we are really going to be busy solving many problems for several years, to realize the technical case of multi-core innovation. Until then , enterprise applications like databases, CRM, .ERP better be run on platforms where multi-core CPUs are used up by the applications, well, with the CPU schedulers that has close knowledge on both CPU topology as well as application execution patterns. Let our next generation data centers be blessed with large scale CMP based servers and let the applications be fluid, fast and highly available with high-density compute servers. Unfortunately an IT manager/ system admin now should be more technical in making the best out of hardware, to keep their application response time lowest possible, And application developers cannot afford to be “mere application architects” as they now have to make applications ready for multi-core world.... which is the biggest challenge.

9. About the Author

Peter Chacko has been working on system/networking/storage systems development since 1994. His career spans Hard-real time/embedded systems, OS kernels like Unix and Linux, distributed storage and networking systems, virtualization, data security and related computer science disciplines. He had been working for companies like Bellcore (Bell labs spin-off), HP, IBM, US West communications and a couple of startups as a consultant in the United states for 6+ years. He holds a Masters degree in Computer science & applications(MCA), Bachelors in Physics. He is currently the founder & CTO of Bangalore-based cloudStorage startup(Sciendix data systems pvt.Ltd). He also runs NetDiox computing systems as a not-for-profit research center-cum-educational center on system and networking software alongside. Prior to this he was working for calsoftlabs heading all the R&D activities of storage networking business unit. His linked-in profile is at <http://www.linkedin.com/pub/peter-chacko/b/608/608>