ISSN: 2393-8528

IJICSE
Open Access

Contents lists available at www.ijicse.in

International Journal of Innovative Computer Science & Engineering

Volume 1 Issue 2; Page No. 24-27

Efficient operating system scheduling for symmetric multi-core architectures in CPU scheduling"

Sunil Yadav

Lecturer, Devendra Singh Institute of Technology & Management, Gaziabad, India

ARTICLE INFO

Received 09 Oct. 2014
Accepted 20 Nov. 2014

Corresponding Author:

Sunil Yadav

1 Devendra Singh Institute of Technology & Management, Gaziabad, India

ABSTRACT

As multi-core architectures begin to emerge in every area of computing, operating system scheduling that takes the peculiarities of such architectures into account will become mandatory. Due to architectural differences to traditional multi-processors, such as shared caches, memory controllers and smaller cache sizes available per computational unit, it does not suffice to simply schedule tasks on multi-core processors in the same way as on SMP systems. It will be the responsibility of the operating system to spare the programmer as much platform-specific knowledge as possible and optimize overall performance by employing intelligent and configurable scheduling mechanisms. In this paper we will discuss about multi core architecture with the help of no. of scheduling mechanisms.

1. INTRODUCTION

In the last few years, multi-core CPUs have become a standard component in nearly all sorts of computers – not only servers and high-end workstations but also desktop and laptop PCs for consumers and even game consoles nowadays usually come with CPUs with more than one core. In order to implement the exponential increase of integrated circuits, the transistor structures have to become steadily smaller. On the one hand, the extra transistors were used for the integration of more and more specialized instruction sets on CISC chips. On the other hand, smaller transistor sizes led to higher clock rates of the CPUs, because due to physical factors, the gates in the transistors could perform faster state switches. However, since electronic activity always produces heat as an unwanted by-product, the more transistors are packed together in a small CPU die area, the higher the resulting heat dissipation per unit area becomes With the higher switching frequency, the electronic activity was performed in smaller intervals, and hence more and more heat-dissipation emerged. The cooling of the processor components became more and more a crucial factor in design considerations and it became clear, that the increasing clock frequency could no longer serve as the primary reason for processor speedup.

1.1 Multi core scheduling:

One could assume that the scheduling process on such multi-core processors wouldn't differ much from conventional scheduling – intuitively the run-queue would just have to be replaced by n run-queues, where n is the number of cores and processes would simply be scheduled to the currently shortest run-queue (with some additional process-priority treatment, maybe).

While that might seem reasonable, there are some properties of current multi-core architectures that speak strongly against such a naïve approach. First, in many multi core architectures, each core manages its own level 1 cache (Figure 1).

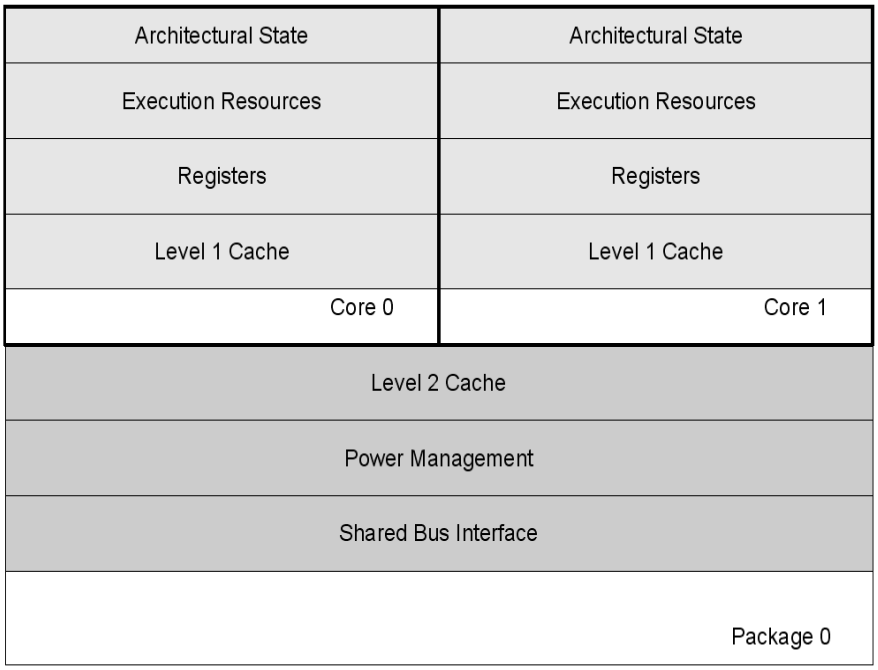

Figure 1: Typical multi-core architecture

2. Operating System Scheduling:

- **Scheduling Domains**: Linux load-balancing takes care of different cache models and computing architectures but at the moment not necessarily of performance asymmetry. The underlying model of the Linux load balancer is the concept of scheduling domains, which was introduced in Kernel version2.6.7 due to the unsatisfying performance of Linux scheduling on SMP and NUMA systems in prior versions The scheduling domain concept introduces scheduling domains, a logical union of computing resources that share common properties, with whom it is reasonable to treat them equally and CPU groups within these domains. Those groups contain hardware-addressable computing resources that are part of the domain on which the balancer can try to even the domain load out.

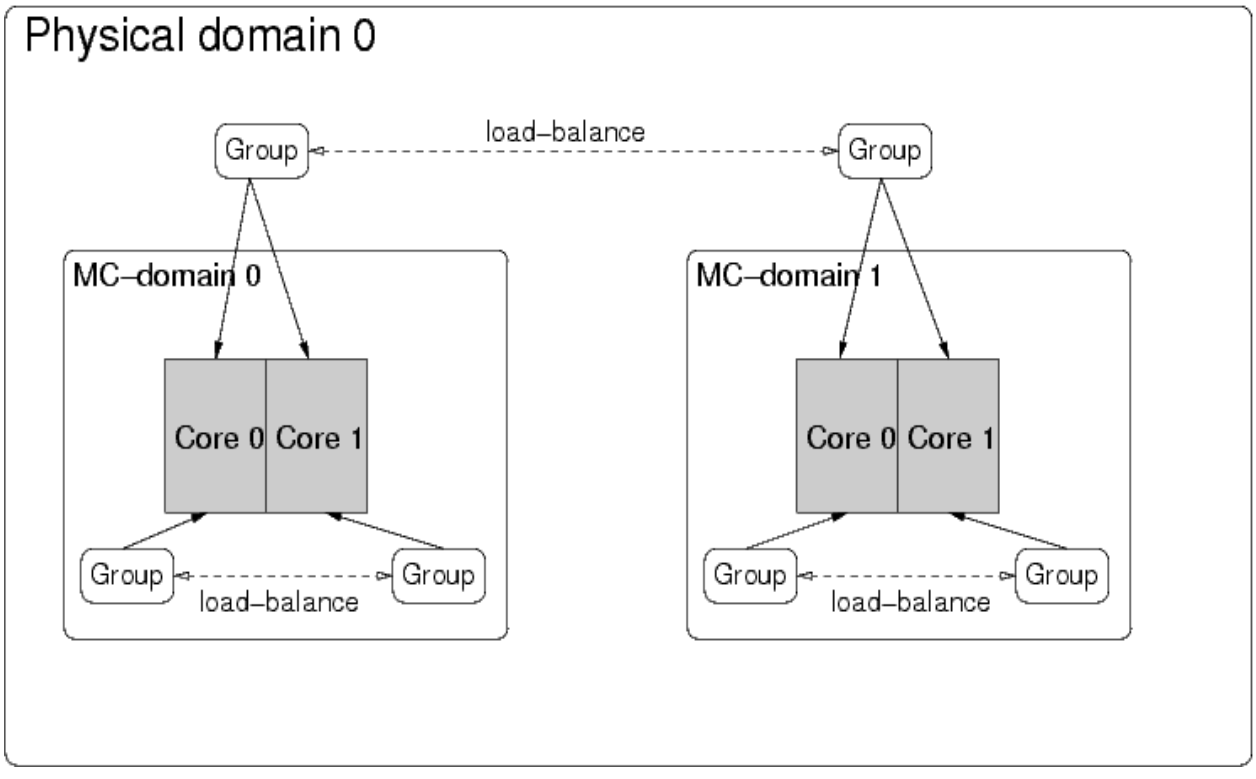

Figure 2: Example hierarchy in the Linux scheduling domains

- Windows scheduler

In Windows, scheduling is conducted on threads. The scheduler is priority-based with priorities ranging from 0 to 31. Time slices are allocated to threads in a round-robin fashion; these time slices are assigned to highest priority threads first and only if know thread of a given priority is ready to run at a certain time, lower priority threads may receive the time slice. However, if higher-priority threads become ready to run, the lower priority threads are preempted. Scheduling on SMP-systems is basically the same, except that Windows keeps the notion of a thread's processor affinity and an ideal processor for a thread. The ideal processor is the processor with for example the highest cache-locality for a certain thread. However, if the ideal processor is not idle at the time of lookup, the thread may just run on another processor.

3. Problem Defination:

Research on multi-core scheduling deals with a number of different topics, many of which are orthogonal (e.g. maximizing fairness and throughput). The purpose of this section is to present an interesting selection of different approaches to multi-core scheduling.

3.1Cache-Fairness

The situation is unsatisfactory due to several reasons: First, it can lead to unpredictable execution times and throughput and second, scheduling priorities may loose their effectiveness because of threads running on cores with aggressive "co-runners" (i.e. threads running on another core in the same package).Figure 3 shows such a scenario: Thread B uses the larger part of the shared cache and thus maybe negatively influences the cycles per instruction that thread A achieves during its CPU time share.

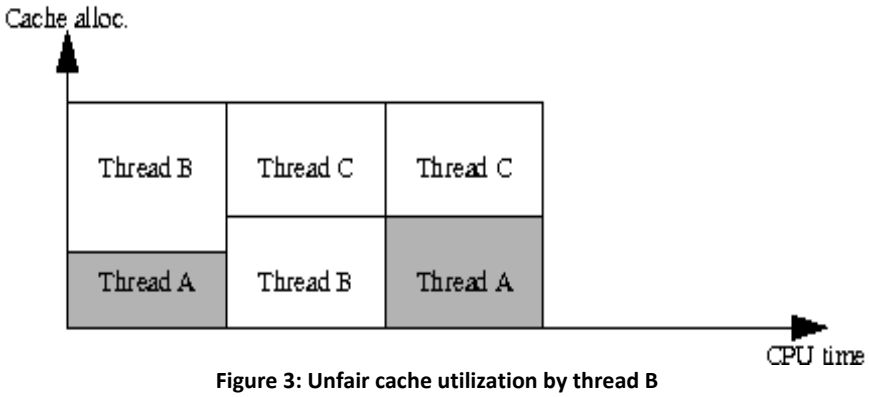

Figure 3: Unfair cache utilization by thread B

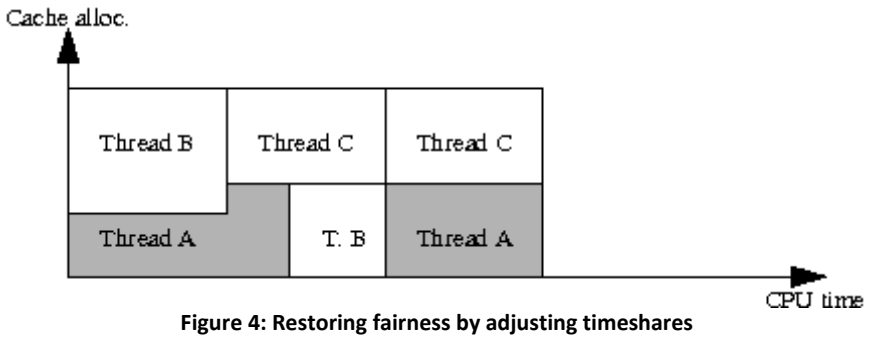

Figure 4: Restoring fairness by adjusting timeshares

3.2 Balancing core assignment:

This benefit function is based on three inputs

Components:

1) The normalized core preference of a thread, which is based on the instructions per Cycle that a thread j can achieve on a certain core i (*j i IPC* ,), normalized by max(*j k IPC* ,) (where k is an arbitrary CPU/core)

2) The *cache-affinity*, a value which is 1 if the thread j was scheduled on core i within a

Tunable time period and 0 otherwise

3) The average cache investment of a thread on a core which is determined by inspecting the hardware cache miss counters from time to time.

3.3 Performance symmetry:

It has been advocated that building multi-core chips with asymmetric performance of the Different cores can have advantages for the overall processing speed of a CPU.

For example:

It can prove beneficial if one fast core can be used to speed up parts that can hardly be parallelized while multiple slower cores come to play when parallel code parts are executed. By keeping the cores for parallel execution slower than the core(s) for serial execution, die area and cost can be saved. While power consumption may be reduced.

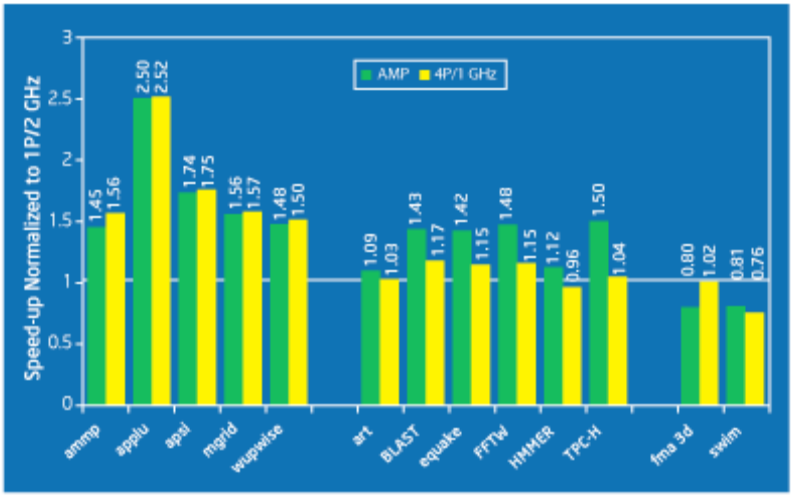

Figure 5: Comparison of speedup with SMP and AMP using highly parallel programs (left), moderately.

4. CONCLUSION:

The probability is high, that processor architectures will undergo extensive changes in order to keep up with Moore's law in the future. AMPs and many-core CPUs are just two proposals for innovative new architectures that may help in prolonging the time horizon within which Moore's law can stay valid. Operating system schedulers are going to have to adapt to the changing underlying architectures. Achieving fairness and repeatability on today's available multi-core architectures are the major design goals of the scheduling techniques detailed in The first approach is justified by a number of experimental results that show that priorities are actually enforced much better than with conventional schedulers; however it remains to be seen.

5. REFERENCES:

1. G. E. Moore: „Cramming more components onto integrated circuits", *Electronics,*Volume 38, Number 8, 1965.
2. O. Wechsler: „Inside Intel® Core™ Micro-architecture", *Intel Technology Whitepaper.*
3. Li et al.: „Efficient Operating System Scheduling for Performance-Asymmetric Multi-Core Architec-tures*", In: International conference on high performance computing, networking, storage, and analysis,* 2007
4. Balakrishnan et al.: „The Impact of Performance Asymmetry in Emerging Multicore Architectures", *In Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 506–517, June 2005.
5. M. Annavaram, E. Grochowski, and J. Shen: "Mitigating Amdahl's law through EPI throttling". In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 298–309, June 2005.
6. V. Pallipadi, S.B. Siddha: "Processor Power Management features and Process Scheduler: Do we need to tie them together?" In: *LinuxConf Europe* 2007.
7. A. Fedorova, M. Seltzer and M. D. Smith: "Cache-Fair Thread Scheduling for Multicore Processors", Technical Report TR-17-06, Harvard University, Oct. 2006
8. S. Kim, D. Chandra and Y. Solihin: "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture", In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques,* 2004
9. D. Menasce and V. Almeida: "Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures", In: *Proceedings of the 4th International Conference on Supercomputing*, June 1990.