System Programming Research Group

Dept. of Computer Science, System and Industrial Engineering

University of Rome "Tor Vergata"

Operating System's Support for Multicore Systems: Current and Future Trends

24/04/2007

E. Betti, D. P. Bovet, M. Cesati, R. Gioiosa

REVISIONS		
Rev	Date	Notes
0	29/03/2007	First version
1	24/04/2007	Minor revision

Introduction

Hardware trends

In the quest for the highest CPU performances, hardware developers are faced with a difficult dilemma. On one hand, the Moore's Law does not apply to computational power any more, that is, computational power is no longer doubling every 18 months as in the past. On the other hand, power consumption continues to increase more than linearly with the number of transistors included in a chip, and the Moore's Law still holds for the number of transistors in a chip.

Several technology solutions have been adopted to solve this dilemma. Some of them try to reduce the power consumption by sacrificing computational power, usually by means of *frequency scaling*, *voltage throttling*, or both. For instance, the Intel Centrino processor [1] has a variable CPU clock rate ranging between 600 MHz and 1.5 GHz, which can be dynamically adjusted according to the computational needs.

Other solutions try to get more computational power from the CPU without increasing power consumption. For instance, a key idea was to increase the *Instruction Level Parallelism* (ILP) inside a processor; this solution worked well for some years, but nowadays the penalty of a cache miss (which may stall the pipeline) or of a miss-predicted branch (which may invalidate the pipeline) has become way too expensive.

Chip-Multi-Thread (CMT) [2] processors aim to solve the problem from another point of view: they run different processes at the same time, assigning them resources dynamically according to the available resources and requirements. Historically the first CMT processor was a *coarse-grained multithreading* CPU (IBM RS64-II [3, 4]) introduced in 1998: in this kind of processor only one thread executes at any instance. Whenever that thread experiments a long-latency delay (such as a cache miss), the processor swaps out the waiting thread and starts to execute the second thread. In this way the machine is not idle during the memory transfers and, thus, its utilization increase.

Fine-grained multithreading processors improve the previous approach: in this case the processor executes the two threads in successive cycles, most of the time in a round-robin fashion. In this way the two threads are executed at the same time but, if one of them encounters a long-latency event, its cycles are lost. More-

over, this approach requires more hardware resources duplication than the coarse-grained multithreading solution.

In *Simultaneous MultiThreading* (SMT) processors two threads are executed at the same time, like in the fine-grained multithreading CPUs; however, the processor is capable of adjusting the rate at which it fetches instructions from one thread flow or the other one dynamically, according to the actual environmental situation. In this way, if a thread experiments a long-latency event, its cycles will be used by the other thread, hopefully without loosing anything.

Yet another approach consists of putting more processors on a chip rather than packing into a chip a single CPU with a higher frequency. This technique is called *chip-level multiprocessing* (CMP), but it is also known as "chip multiprocessor"; essentially it implements symmetric multiprocessing (SMP) inside a single VLSI integrated circuit. Multiple processor cores typically share a common second- or third-level cache and interconnections.

In 2001 IBM introduced the first chip containing two single-threaded processors (cores): the POWER4 [5]. Since that time, several other vendors have also introduced their multicore solutions: dual-core processors are nowadays widely used (e.g., Intel Pentium D [6], AMD Opteron [7], and Sun UltraSPARC IV [8] have been introduced in 2005); quad-core processors are starting to appear on the shelves (Intel Pentium D [9] was introduced in 2006 and AMD Barcelona will appear in late 2007); eight-core processors are expected in 2008.

In 2005 IBM, in collaboration with Sony and Toshiba, introduced the Cell Broadband Engine (CBE) microprocessor [10, 11]. Cell is a heterogeneous chip multiprocessor that consists of a 64-bit PowerPC core and eight specialized SIMD co-processors called Synergistic Processor Units (SPU). The cores are connected by means of a on-chip bus. The Cell processor is aimed at data-intensive processing, like that found in cryptography, media, and scientific applications. The Cell processor is the main computational unit of the Sony's PlayStation 3 game computer, thus it is expected that IBM will invest significant efforts and resources in this architecture.

Multi-core chips will become ubiquitous in the next few years. It has been foreseen [12] that in a near future even many embedded systems will sport multicore chips, because the small increase in power consumption will likely be justified by the large increment of computational power available to the embedded system's applications. Furthermore, the actual trend in the design of system-on-chip devices suggests that in a near future such chips will include multicore processors.

Therefore, the embedded system designers will be able to create boards having many processors almost "for free", that is, without the overhead of a much more complicated electronic layout or a much higher power consumption.

OS support for multicore chips

Operating systems must evolve in order to fully support the new multithreading and multicore CPUs. Current operating systems, in fact, basically consider each virtual processor or omogeneous core as a separate, independent CPU—these systems are thus handled like classic multiprocessor platforms. However, the OS does not reckon that virtual processors and cores often share critical hardware resources inside a chip. Therefore, the OS may fail in fully exploiting the capabilities of the system.

For instance, in a system sporting two dual-core chips and running two CPU-bounded processes, the OS may end up assigning the processes to the two cores on the same chip, while the cores on the other chip remains idle: because the cores in the same chips contend for some resources—hardware caches, for instance—the two processes are not executed at the maximum speed that the system would allow.

For another instance, if the OS would be aware of the peculiar hardware characteristics of each processing unit in the system, it could schedule the active processes in such a way to achieve the best compromise between required performances and system's power consumption.

Heterogeneous multicore chips lead to even bigger problems. At the present, no operating system at all natively supports heterogeneous cores. This means that user applications must directly program these cores, without any help from the operating system. It is not surprising that programming these heterogeneous cores is a tedious, error-prone, and heavy job.

ASMP-LINUX

ASMP-LINUX, *ASymmetric MultiProcessor Linux*, is an extension of the Linux® operating system kernel that can be used in multiprocessor and multicore systems

with hard real-time requirements. It has been originally developed as a patch for the 2.4 Linux kernel series in 2002 [13]. After several revisions and major updates, it is now implemented as a patch for the Linux kernel 2.6.19.1. ASMP-LINUX is released under the version 2 of the GNU General Public License [14], and it is available to all developers who wish to work with it.

ASMP-LINUX provides real-time capabilities while maintaining the software architecture relatively simple. In a conventional (symmetric) kernel, I/O devices and CPUs are considered alike, since no assumption is made on the system's load. Asymmetric kernels, instead, consider real-time processes and related devices as privileged and shield them from other system activities.

The main advantages offered by ASMP-LINUX to real-time applications are:

- Deterministic execution time (up to a few hundreds of nanoseconds).
- Very low system overhead.
- High performance and high responsiveness.

One of the design goals of ASMP-LINUX is simplicity: because Linux developers introduce quite often significant changes in the kernel, it would be very difficult to maintain the ASMP-LINUX patch if it would be intrusive or overly complex. Actually, most of the code specific to ASMP-LINUX is implemented as an independent kernel module, even if some minor changes in the core kernel code are still required.

Another design goal of ASMP-LINUX is architecture-independency: the patch can be easily ported to many different architectures, besides the IA-32 architecture that has been adopted for its first implementation.

ASMP-LINUX is a vertically partitioned operating system; it implements two different kinds of partitions: *system partitions* and *real-time partitions*. The system partition executes all the non real-time activities, such as daemons, normal processes, interrupt handling for non critical devices, and so on. Each real-time partition handles some real-time tasks, as well as any hardware device and driver that is crucial for the real-time performances of that tasks.

In an ASMP-LINUX system there is exactly one system partition, which may consist of several processors, devices, and processes; moreover, there should always

exist at least one real-time partition. Additional real-time partitions might also exist, each handling one specific real-time application.

Each real-time partition consists of a processor (called *shielded CPU*, or shortly S-CPU), $n_{irq} \geq 0$ IRQ lines assigned to that processor and corresponding to the critical hardware devices handled in the partition, and $n_{task} \geq 0$ real-time processes (there could be no real-time process in the partition; this happens when the whole real-time algorithm is coded inside an interrupt handler). The real-time partition is protected from any external event or activity that does not belong to the real-time task running on that partition. Thus, for example, no conventional process can be scheduled on a shielded CPU and no normal interrupt can be delivered to that processor.

To validate the claim that ASMP-LINUX provides a good foundation for an hard real-time operating system on multiprocessor systems, we performed some experiments aimed to measure the operating system overhead and latency of ASMP-LINUX on a few typical multiprocessor platforms, including coarse-grained multithreading processors and multicore chips. The test results show that ASMP-LINUX running on multi-processor and multicore machines is capable of minimizing both operating system overhead and latency, thus providing deterministic results for the tested applications.

A proposal for future researches

High performance multicore processors—particularly the heterogeneous ones—are very complex chips, which often require entirely new programming models and new optimization techniques. Therefore, programming these chips is, in general, an hard task [15].

We propose to adapt the Linux operating system kernel in such a way to natively support homogeneous and heterogeneous multicore chips so as to:

- offer to the programmer an easy, general programming model, regardless of the hardware characteristics of the cores
- increase the system's overall performances by exploting the computational power of specialized cores

- reduce the system's power consumption when the system is running on low-charged batteries or the system load is low

ASMP-Linux for homogeneous and heterogeneous multicore chips

In order to achieve these goals, we plan to extend the basic concepts of ASMP-LINUX. The new operating system will be based on Linux and will be open source, available for the scientific community.

We think that ASMP-LINUX is a good starting point because of its notion of "vertically partitioned hardware resources": the user may freely change the logical processors included in each partition, as well assign processes and IRQ lines—that is, hardware devices—to any partition.

However, the current interface of ASMP-LINUX is cumbersome and not very intuitive. Since all the system programmers know how to use resources such as files, serial ports, audio cards, and so on, the first change to ASMP-LINUX will consist in making the hardware resources included in the partitions available as *general resources*, that is, through the well-established POSIX *file interface*. Thus, all programmers who know how to use POSIX file operations will also be able to exploit the cores in their processors.

Proposed implementation

When a real-time process is assigned to a real-time ASMP-LINUX partition, it runs there until it is removed from that partition. Our proposal consists of extending the concept of "real-time process running on an asymmetric real-time partition" to "critical procedure executed on an specific partition".

The system is partitioned in one *system partition* and one or more *specialized partitions*, where each partition may consists of several cores. The cores in the specialized partitions are seen as *resources* capable of executing some specific tasks. The operating system is aware of the available resources and it is in charge of managing the resource pool of each specialized partition. The operating system also handles the data communication channels between different cores.

A user application can request a resource included in some specialized partition where a specific procedure shall be executed. In order to do this, the process requests the operating system a *service*; the operating system will take the pending request—consisting of both the procedure code and the input parameters—and will forward it to the best resource it has currently available in the partition's pool. Then, the process can wait for the procedure to complete (blocking operation) or it can continue its execution while its request is being processed. In the latter case, the operating system will notify the process as soon as the procedure running on the specialized partition has completed.

Since the number of requests could be greater than the number of resources (cores) in a specialized partition, the operating system implements a queue mechanism similar to the Linux's *work-queue*: when a new request comes, it is put into the proper queue and served according to the queue policy (for example FIFO or predefined priority).

Examples of applications

A typical example is a system where one specialized partition includes cores aimed at vector-matrix multiplications. In this case, the procedures to be executed on the specialized partition will likely be code fragments operating on vectors and matrices. Thus, the process that is executing a user application remains on the system partition, but it can require the execution of specific procedures on a specialized core. Once the procedure has been completed, the output results will be returned to the calling process.

A specialized partition does not necessarily include heterogeneous cores: it might be composed of general-purpose cores similar to the cores in the system partition. The cores in this specialized partition might be used whenever the system load increases, or when the user application must perform some kind of recurring, costly operations.

For instance, whenever a user application allocates a new page of memory, the operating system kernel must fill the page frame with zeros—otherwise the process might get access to information owned by the process that previously owned the page. Filling a page with zeros is a costly task, because it evicts from the hardware caches a lot of cache lines. By demanding this task to a general-purpose core in a specialized partition, the operating system may preserve the hardware cache lines of the core in the system partition, thus potentially achieving higher

performances.

Dynamic behaviour

A distinct characteristic of our proposed operating system extension is that the cores can be dynamically added or removed in the partitions.

In some cases, in fact, it does not pay off to have many cores for specialized functions. For instance, when the system load is high but few processes (or none at all) need services from the specialized partitions, the specialized cores are mostly idle, while the system cores are overloaded. In this case, if all cores are homogeneous, it is convenient to remove some cores from the specialized partitions and add them to the system partition, so that the overall performances of the system increase. In the extreme case in which all specialized partitions are empty (no core assigned), the system becomes a classical Symmetric MultiProcessor (SMP) platform.

Another typical case is when the system load is very low: some core can be removed from a partition and turned off, thus saving power. As soon as the system load increases, the operating system may decide to turn some core on and to add it to the proper partition, according to the kind of the most required services.

In general, some cores might be turned off when there are few requests for the kind of services they provide. This means that, in heterogeneous systems, a specialized core could be turned off even if the system load is high, because few processes need the service provided by the core. In fact, switching the core off would not significantly affect overall performances but, at the same time, it will decrease system's power consumption.

However, is always up to the operating system to determine if and when turning a core on or off, or moving homogeneous cores from a specialized partition to the system partition, and so on. The correct choice will be taken at run-time looking at the overall system statistics kept by the operating system (for example the number of requests in each partition's work-queue), in a way completely transparent to the user.

Interface

As we explained, our extension to the Linux operating system kernel will provide a I/O POSIX-like interfaces.

Specifically, some *device files* will be available for the user, where each device file will represent a specialized partition. The user can interact with a specialized partition through the regular POSIX file operations:

open Notify the operating system that the process might request some services from the specialized partition. This operation is like a *registration*. The operating system will use the number of processes registered for using the services of a specialized partition in order to dynamically adjust the number of cores in the partition.

write Post a service request to the specialized partition.

read Obtain the output data produced by the procedure, or just check whether the procedure has been completed.

poll Wait sinchronously until an event associated with a number of partitions occurs—most likely, the termination of a procedure running on a specialized core.

close De-register the process from using the partition. If no process is using the partition, the operating system may remove all the cores from the partition, thus shutting the corresponding services down.

ioctl Implement all the features with a semantic that does not fit in the previous operations (such as forcing the kernel to send a signal to the process when a procedure running on a specialized core terminates).

Road map

Because the targets of this proposal are both homogeneous and heterogeneous multicore chips, we plan to organize the future research work in two phases.

In the first phase, ASMP-LINUX will be modified so that it will be able to handle the cores in the specialized partitions as resources available for the processes.

This will be done on homogeneous multicore processors, so that it will not be necessary to deal with different instruction sets or overly complex communicational channels among the cores. Several homogeneous multicore processors are already available on the market, thus this activity can start immediately.

In the second phase, the work done in the first phase will be adapted for heterogeneous multicore processors. We are expecting that most of the code developed in the first stage will be architecture-independent, thus the main job in this phase will consist in writing an architecture-dependent layer below the layer produced in the first phase.

In order to understand how to write the architecture-dependent layer, some skills on the real processors will be required. This activity can be done in parallel during the first phase. Currently, the most powerful and widely available heterogeneous multicore chip present on the market is the IBM Cell processor, thus we will focus on this architecture.

However, one of our goals will be to make the porting the architecture-dependent layer to new hardware as easier as possibile.

References

[1] Intel Corp., "Intel Core2 Duo Mobile Processor datasheet," 2006. Available from `http://download.intel.com/design/mobile/datashts/31407801.pdf`.

[2] H. M. Mathis, A. E. Mericas, J. D. McCalpin, R. J. Eickemeyer, and S. R. Kunkel, "Characterization of simultaneous multithreading (SMT) efficiency in POWER5," *IBM J. Res. Dev.*, vol. 49, no. 4/5, pp. 555–564, 2005.

[3] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel, "A multithreaded PowerPC processor for commercial servers.," *IBM Journal of Research and Development*, vol. 44, no. 6, pp. 885–898, 2000.

[4] S. Storino, R. Eickemeyer, R. Kalla, and S. Kunkel, "A commercial multithreaded RISC processor," *Digest of Papers, International Solid-state Circuits Conference*, pp. 236–237, 1998.

[5] J. M. Tendler, J. S. Dodson, J. S. F. Jr., H. Le, and B. Sinharoy, "POWER4 system microarchitecture," *IBM Journal of Research and Development*, vol. 46, no. 1, pp. 5–26, 2002.

[6] Intel Corp., "Intel Pentium D Processor 900 sequence and Intel Pentium Processor Extreme Edition 955, 965 datasheet," 2006. Available from http://download.intel.com/design/PentiumXE/datashts/31030606.pdf.

[7] Advanced Micro Devices, "AMD Opteron™ Processor Product Data Sheet," 2006. Available from http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/23932.pdf.

[8] Sun Microsystems, "UltraSPARC® IV Processor Architecture Overview," Feb. 2004. Available from http://www.sun.com/processors/whitepapers/us4_whitepaper.pdf.

[9] Intel Corp., "Intel Quad-Core processors." Available from http://www.intel.com/quad-core/index.htm.

[10] M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "A novel SIMD architecture for the Cell heterogeneous chip-multiprocessor," *Hot Chips*, vol. 17, Aug. 2005. Available from http://www.hotchips.org/archives/hc17/2_Mon/HC17.S1/HC17.S1T1.pdf.

[11] M. Gschwind, P. Hofstee, B. Flachs, M. Hopkins, Y. Watanabe, and T. Yamazaki, "Synergistic processing in Cell's multicore architecture," *IEEE Micro*, Mar. 2006. Available from http://www.research.ibm.com/people/m/mikeg/papers/2006_ieeemicro.pdf.

[12] P. McKenney, "SMP and embedded real-time," *Linux Journal*, vol. 153, Jan. 2007. Available from http://www.linuxjournal.com/article/9361.

[13] R. Gioiosa, "Asymmetric kernels for multiprocessor systems (in Italian)," October 2002. Master thesis, University of Rome "Tor Vergata".

[14] Free Software Foundation, Inc., "GNU General Public License, version 2," June 1991. Available from http://www.gnu.org/licenses/gpl2.html.

[15] D. Scarpazza, O. Villa, and F. Petrini, "Programming the Cell processor," *Dr. Dobb's Journal*, pp. 26–30, Apr. 2007.