Multithreaded Architectures

Filipe José Silva de Campos

Departamento de Informática, Universidade do Minho
4710 - 057 Braga, Portugal
Filipe_campos@yahoo.com

Abstract. The increasing gap between CPU and memory speed are driving several optimisations to be used in the architecture of novel processors, such as the hyper threading technology. This communication explores this technology, and also attempts to draw some conclusion about its real value.

1 Introduction

For the past several years, the key to increase performance used by the developers is higher clock frequencies. But now, current microprocessor industry struggles with scaling development and design costs. Researchers have begun to realize the increases of complexity and speed of the processor as the effect to increase the cost with no guarantee that such designs will meet significant additional performance gains. The large gap between the processor frequency and the memory speed is having effect on the portion of execution time wasted by the processor on cache misses. Memory latency, the time required to initiate, process, and return the result of a memory request, has always been the bane of high-performance computer architecture, and it is especially critical in large-scale multiprocessors. Many processor cycles may elapse while a request is communicated among physically remote modules.

Consequently, researchers have created several optimisations to minimize this problem. The first of these is the use of multithreading to enable the processor to perform useful instructions during cache misses. This provides a way to create a latency tolerance, increasing the global performance of the processor, without increasing the complexity and costs of the chip.

This communication makes a study of a multithreaded processor model (hyper threading processor), describing its architecture, performance, and making some conclusions about its real value.

2 Multithreaded Architectures

Multithreaded processors aim to reduce the inefficiencies in the processor due to operational latencies, such as cache misses or instruction with long execution cycles. A conventional single threaded processor will wait during a remote reference, so we may say is stopped for a period of time. A multithreaded machine, on the other hand, will suspend the current context and switch to another, so after some fixed number of cycles it will again be working doing useful work. The objective is to maximize the fraction of time that the processor is busy, given by the next equation:

$$\text{Efficiency} = \frac{Working}{Working + Switching + Stopped}$$

where *Working*, *Switching*, and *Stopped* represent the amount of time, measured over some large interval, where the processor is in the corresponding state. The basic idea behind a multithreaded machine is to interleave the execution of several contexts in order to reduce the value of *Stopped*, but without overly increasing the value of *Switching*. Because the processor can switch contexts, it is necessary to maintain separate thread states using independent program counters and registers for each thread. Since this processor have separate contexts and instruction sources, a multithreaded architecture is capable of taking advantage of thread-level parallelism (TLP).

A processor that can support several active threads of computation can be classified as a multithreaded architecture; two models are currently being explored:

Multiprocessor architecture. It consists of multiple processor cores in a single package or two or more physical processor in a single computer. This improves overall performance by allowing threads to execute in parallel**.** Like multithreaded processors, multiprocessor architectures rely on thread-level parallelism to improve performance. However, this architecture does nothing to improve processor resource utilization because a processor's resource will go unused when not enough parallelism can be found in the thread being used.

As show in Figure 1, vertical waste is introduced when the processor issues no instruction in a cycle, and horizontal waste when not all issues slots can be put in a cycle.

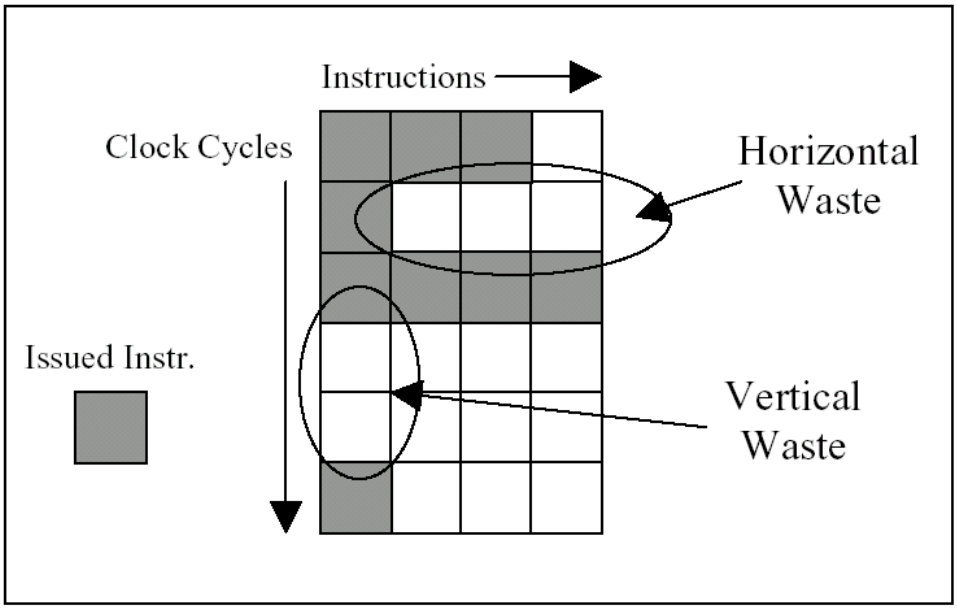

Figure 1: Empty slots can be defined as vertical waste or horizontal waste.

Simultaneous Multithreading (SMT) improves the performance in two ways. First, this architecture takes simultaneous advantage of thread and instruction–level parallelism (ILP). This allows the processor to execute instructions from multiple threads within a single clock cycle, eliminating the horizontal waste. Second, if only one thread is active due to long-latency, that thread may consume all of the available issue slots. As a result, vertical waste is eliminated due to the availability of unblocked instructions in other threads.

In an SMT processor, on each cycle, instructions are selected for execution from all threads.

3 Case Study: Hyper Threading

In mid-2002, Intel introduced an updated version of the Xeon line of processors that includes simultaneous multithreading. They have chosen to brand this technology “Hyper - Threading”. Like the theoretical SMT processors, multiple architecture states are maintained, thus Hyper-Threading allows a single physical processor appear as more than one “logical” processor. Hyper threading processor is a multithreaded architecture that uses only one processor and supports two threads (2 contexts). It switches utilization of the chip resources from the currently executing thread to a second thread, when the currently executing one initiates a long latency operation. This reduces the likelihood of long pipeline stalls by allowing the second thread to execute while the long latency operation of the first thread completes. For example, one logical processor can execute a floating-point operation while the other logical processor executes an addition and a load operation.

3.4 Architecture Implementation

To operate two threads with a single processor, the architecture state (AS) of each thread must be maintained in the hardware. The state of a thread consists of the GPRs, FPRs, condition register (CR), count register, link register, fixed-point exception register, and floating-point status and control register. All of the registers specifically listed above are replicated in order to establish a complete and private state set. None of the replicated facilities are very large; in fact, both threads share all the largest parts of the design, such as the caches and ALU. The area added in the chip is less than 5 %, and having very little impact on cycle time.

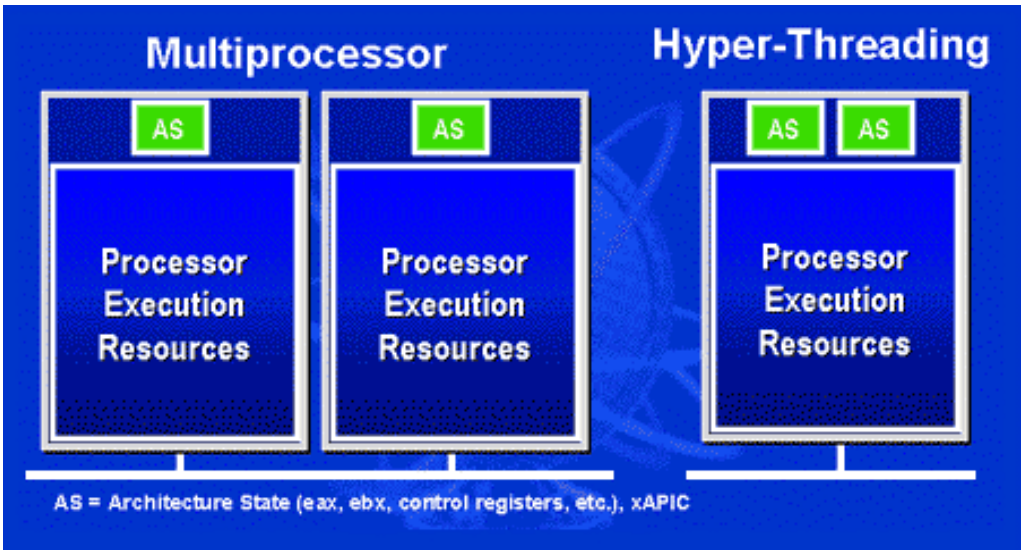

Figure 2: Architecture of hyper threading

In the Xeon processor with Hyper-Threading technology, logical processors share and separate the following functionality. However, to satisfy the need of performance when only one thread is active, the partitioned resources are capable of being recombined when multiple threads do not exist.

Shared	Separate
Decode logic	Interrupt controller
Caches	Next instruction pointer
Execution units	Instruction TLB's
Branch predictors	Return stack buffer
Control logic	Register alias table
Busses	

To support hyper threading it is necessary to include some unique registers in the chip. The designer added control registers, such as the thread state control - to specify the condition under which a thread switch can occur –a register to specify single/hyper thread modes of execution and the thread-switch timeout register. This register contains a threshold for the number of cycles that can occur between thread switches. Most of thread switching is the result of cache misses; if a thread stays a long time without a miss, the other thread remains stopped for a long time. This register is used to force the switch of a thread after a specified number of cycles, to prevent one thread from using all the resources.

Pipeline. The processor decodes architectural instructions into micro-operations or "μops." Figure 3 shows a simplified version of the pipeline, where the execution engine includes several mores stages than they are show in the figure.

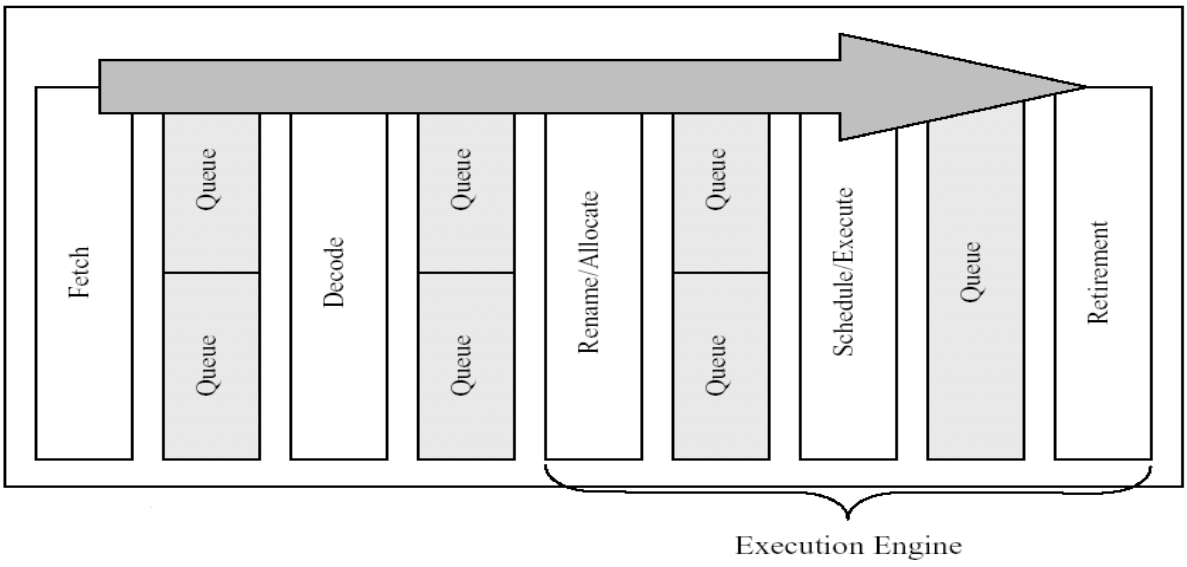

Figure 3: Pipeline of Xeon processor

In the first stage, instructions are fetched, converting instructions into micro-operations or called "μops". Each "μop" is tagged with the thread identifiers and stored in the trace cache. The trace cache is an associative cache that replaces instructions based on a least-

recently-used algorithm. If the two logical processes compete for the trace cache in a single cycle, access is granted to one on the current cycle then the other process on the following cycle. The processor branch prediction hardware, instruction pointer and decode stage buffers are duplicated for each logical processor. The instruction translation lookaside buffer (ITLB) sets the next-instruction pointer based on instructions delivered from the trace cache.

Each logical processor has its own set of two 64-byte streaming buffers to hold instructions prepared for the decode stage. The decode logic is shared but should not be utilized very often due to "μop" translation in the fetch stage. Between the decode logic and the execution engine instructions are queued in two equally sized partitions, ensuring independent forward progress during stalls in the fetch and decode stage. Queued instructions enter the execution engine where physical resources are allocated as needed by each "μop". The Xeon processor has 126 re-order buffer entries, 128 integer physical registers, 128 floating-point registers, 48 load buffers and 24 store buffers. The re-order buffer entries, load buffers and store buffers are partitioned such that each logical processor may only access half the resources. The Register Alias Table (RAT) is used to track mappings between architectural registers and physical registers. Because each logical processor needs to maintain its own state, there are two RATs, one for each logical processor.

The instruction scheduler is shared. It chooses instructions regardless of the logical processor they are associated with. However, the scheduler does make one accommodation; there is a limit on the number of active entries that a logical processor can have in the scheduler's queue. All execution units are shared – this means that they do not require adaptation from the single context Xeon design. Retirement of instructions is done in program order. The retirement process alternates between logical processor threads, looking for operations that are ready to be retired. If there are no operations for one of the two logical processors, all retirement resources are dedicated to the active context.

Memory and Cache. Access to the memory subsystem requires very little enhancement to support hyper-threading. The Data Lookahead Translation Buffer (DTLB) has been enhanced to include a thread identifier for each entry. Additionally, each logical processor has its own reservation register to ensure fairness in processing DTLB misses. As was discussed earlier in this document, because logical processors share memory access resources (caches), there are an increased potential for cache and resource conflicts.

Bus. Bus resources are shared, but each logical processor in the Hyper-Threaded Intel Xeon includes its own interrupt controller. Accesses are treated on a first come, first served basis. To support debugging, the logical processor ID is carried on all bus requests.

4 Benchmarks

The machine used in the test the gains of the hyper threading technology was a Dell Precision 530 Workstation, with two 2.4GHz Xeon processors. It is possible to enable and disable the hyper threading feature and the use of one or two processor in the BIOS.

The configurations used in the tests are:

- 1 CPU, HT Off: one physical logical processor, the result is one logical processor.

- 1 CPU, HT On: one physical processor and hyper threading, the result is two logical processors.
- 2 CPU, HT Off: two physical processors, the result is two logical processors.
- 2 CPU, HT On/2L: two physical processor and hyper threading; two logical processors forcing the machine to use only the first logical processor on each physical processor.
- 2 CPU, HT On/4L: two physical processors and hyper threading, the result is four logical processor.

The X-Axis has abbreviations for the SiSoft Sandra benchmarks software, and represents test arithmetic (ALU[1], FPU[2], FPU-SSE[3]) and multimedia (iSSE[4], fSSE[5]).

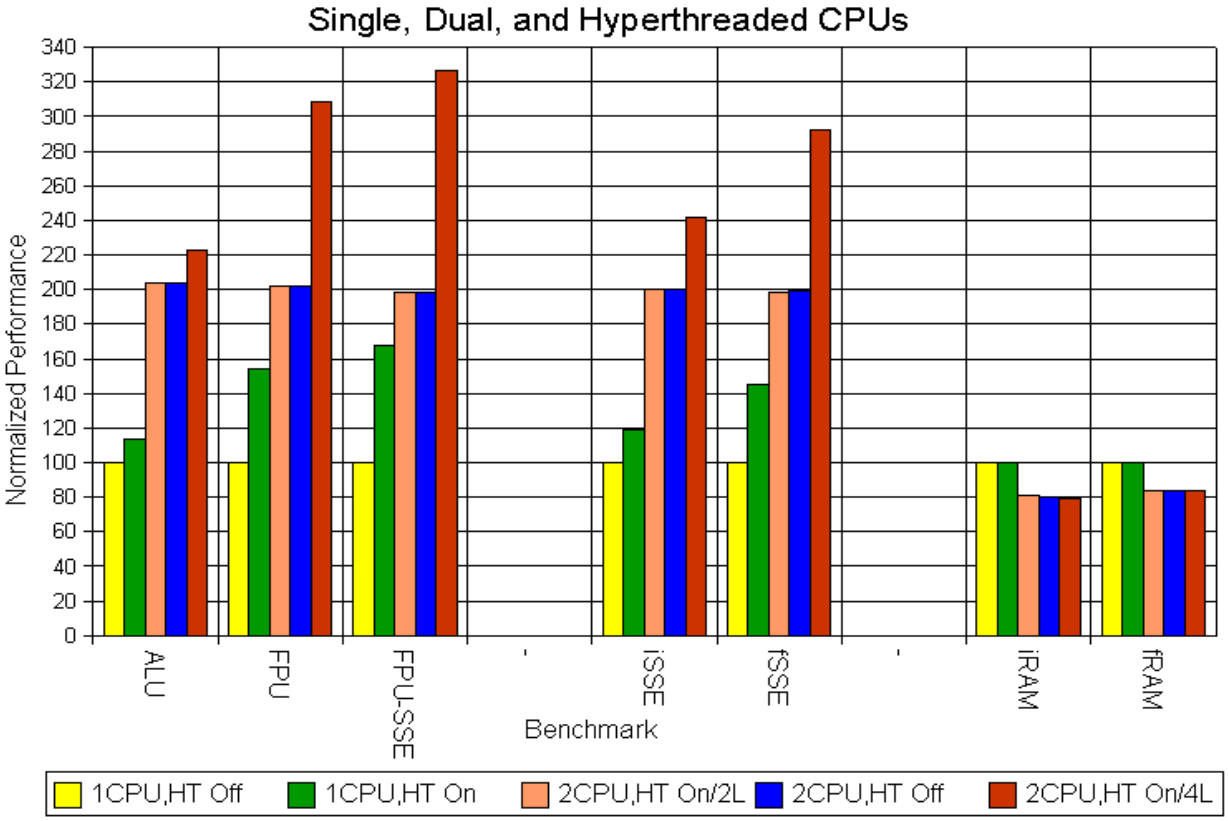

Figure 4: Benchmarks of the Dell Precision 530 Workstation using the SiSoft Sandra benchmarks software.

[1] Dhrystone ALU MIPS
[2] Whetstone FPU MFLOPS
[3] Whetstone iSSE2 MFLOP
[4] Integer iSSE2 it/s
[5] Floating-Point iSSE2 it/s

5 Benefits vs. Cost

5.1 Benefits

Multithreading has been shown to provide a significant performance improvement. Threads can make a GUI more responsive. They can also facilitate the overlap of I/O and computation. If multiple processors are available, threaded applications may see substantial speedup. This is achieved with minimal additional costs in terms of area, cycle time. The chip area added to support multithreading was less than 5%, required primarily for the second set of SPRs, control logic to support thread switching, and added bandwidth in the storage control unit to handle extra outstanding misses. Cycle time was not directly affected by multithreading. The only effect multithreading had on cycle time was additional capacitive wire loading on some resources to support alternate thread paths (less than 1% impact). No additional logic gate levels were introduced in critical paths for multithreading.

5.2 Cost

But, to have this gain it is necessary to use multiprocessor software applications, increasing the complexity and cost to developing this type of applications. For example, the sharing of resources, such as global data, can introduce common parallel programming errors such as storage conflicts and other race conditions.

The main problem with hyper threading is that if one of the two threads on the processor uses up all the CPU's resources (e.g. the thread is in some sort of waiting cycle), it will hamper the other thread's execution. This could lead to a reduction in performance over a standard processor. In a standard processor the operating system would be in charge of deciding which instructions from which threads to run and when and may be able to do a better job of it.

6 Conclusions

Two threads are better than one? The answer is yes. The hyper threading processor considers two separate logical processors on which the software can run. Each logical processor independently responds to interrupts. The first logical processor can track one software thread, while the second logical processor tracks another software thread simultaneously. Because the two threads share the same execution resources, the second thread can use resources that would be otherwise idle if only one thread was executing. This results in an increased utilization of the execution resources within each physical processor.

However, for non-computationally intensive applications and non-multiprocessor software, such as spreadsheets, word processing, or e-mails, which represent the vast majority of applications used by the normal user. The use of hyper threading will not make performance gains.

The use of this technology can increase the performance, but not currently. It may be some case that the performance decreases due to non-optimised software, not designed for parallel computers.

References

[1] J. M. Borkenhagen, R. J. Eickemeyer, R.N. Kalla, and S.R. Kunkel, “A multithreaded PowerPc processor for commercial servers”, November 2000.

[2] Rafael H.Saavedra-Barrera, David E. Culler, and Thorsten von Eicken, “Analysis of Multithreaded Architectures for Parallel Computing”, University of California.

[3] Yen-Kuang Chen, Matthew Holliman, EricDebes, and Sergey Zheltov, “Media Applications on Hyper-Threading Technology”, Intel Labs.

[4] James E Smith, “Instruction-level Distributed Processing”, University of Wisconsin-Madison.

[5] Gurindar S. Sobi, and Amir Roth, “Speculative Multithreaded Processors”, University of Wisconsin-Madison.