32 Hyper-Threading on SMP Systems

If you have not read the book (*Performance Assurance for IT Systems*) check the introduction to "More Tasters" on the web site http://www.b.king.dsl.pipex.com/ to understand the scope and objectives of tasters.

The *CPU Basics* taster, which describes the fundamental issues and solutions that surround CPU performance, is a prerequisite to this topic. It would also be useful to read the taster on SMPs, *Multiprocessors (Shared Memory).* This taster discusses the relatively recent introduction of SMT (Simultaneous Multithreading) techniques to increase CPU throughput. It focuses on the Intel implementation, termed hyper-threading, and primarily limits itself to the implications for SMP systems. A future version will include other implementations of SMT.

32.1 Starting Terminology

CMP (Chip Multiprocessors) is a term that is often used when SMT is discussed. It is the technique where multiple physical processor cores (typically two at the current time) reside on a single chip.

Die is an area of silicon that contains integrated circuits, usually synonymous with a chip.

Hyper-Threading is the name given by Intel to its implementation of SMT on Xeon and Pentium 4 processors.

MT (Multi-Tasking) mode is the term used to describe the hyper-threading mode of operation.

SMP (Symmetric Multiprocessors) is the term that is applied to a system that contains multiple CPUs that are controlled by a single operating system instance, sometimes also known as a tightly-coupled system.

SMT (Simultaneous Multithreading) is the generic term that is used to describe the ability to execute multiple instructions belonging to different threads in a single CPU cycle.

ST (SingleTasking) mode. If the operating system decides that there is insufficient work to warrant hyper-threading it can switch to ST mode which allows a CPU to function as a normal single-threaded CPU, making use of all the resources and operating at full power.

32.2 Technology Outline

32.2.1 Background

The *CPU Basics* taster describes the main techniques that are used to keep a CPU busy by minimising delays such as waiting for memory accesses or pipeline stalls that occur when the next instruction is not in the pipeline, typically due to a branch. The techniques include: processor caching; branch prediction; and parallel instruction execution within a single thread where multiple instruction execution units are available. Despite these optimisations there can still be appreciable delays. They are sometimes categorised as horizontal and vertical

waste; horizontal waste is defined as occasions when there are insufficient independent instructions available for execution to make use of parallel instruction execution within a single thread; while vertical waste covers latency delays, e.g. when waiting for a main memory access to complete.

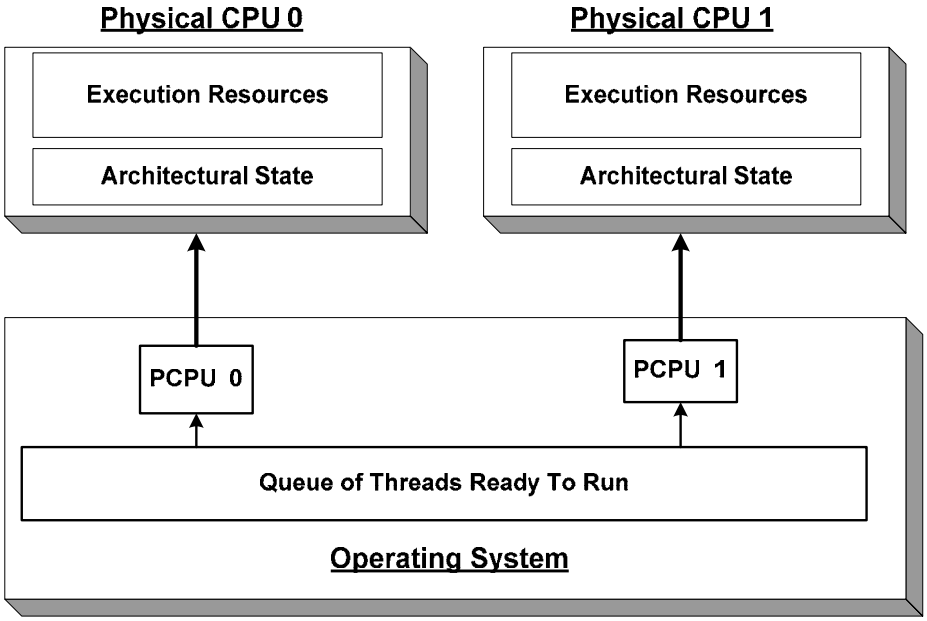

Figure 32-1 High Level View of Dual CPU System

The obvious route to improved throughput is to configure multiple CPUs, typically in an SMP system, as shown in Figure 32-1. See the *Multiprocessing (Shared Memory)* taster for further information. The delays still occur in each CPU but performance is improved by the ability of the operating system to distribute work over multiple CPUs. A recent refinement to SMP configurations that is offered by a number of hardware vendors is to the ability to use CPU chips that consist of two processor cores on a single die, sometimes called CMP (Chip Multiprocessors). As shown in Figure 32-2, the cores include dedicated pipelines, execution units, level 1 data and instruction caches *et al*, everything in fact except the level 2 processor cache which is shared across the cores.

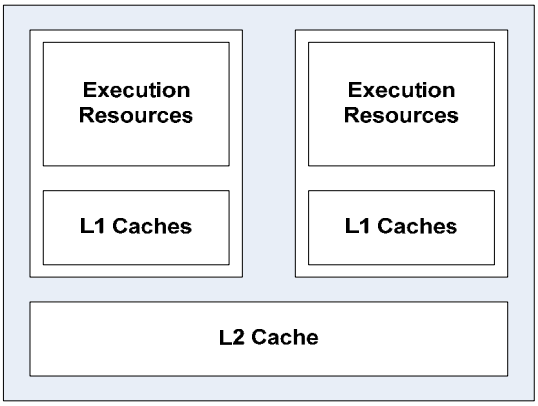

Figure 32-2 Multiprocessor Chip

The latest approach that has been brought to market is SMT (Simultaneous Multithreading) where the objective is to make better use of each individual CPU by supporting the execution

of parallel instructions from different threads. Intel provided the first implementation on its Xeon and Pentium 4 processors under the soubriquet “hyper-threading”.

32.2.2 Hyper-Threading Concept

One of Intel’s key objectives was to minimise any increases in the size of a die that would be required to support the implementation; the actual increase is of the order of 5%. Naturally, this objective influenced the design. Intel’s conceptual diagram, as shown in Figure 32-3, shows that each physical CPU has multiple “architecture states”, currently limited to two. Each architecture state comprises: general purpose registers; control registers; programmable interrupt controller registers; and some machine state registers. The majority of other resources are shared. From a software perspective, each architecture state becomes a logical CPU to which the operating system can dispatch work. The diagram shows a dual CPU system with four logical CPUs.

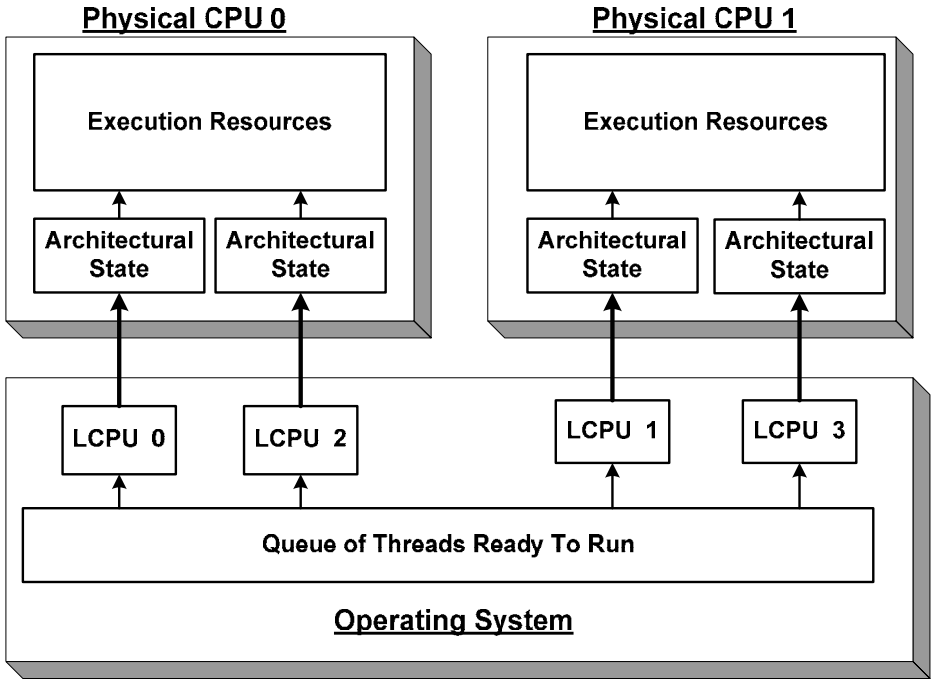

Figure 32-3 Overview of Hyper-Threading on a Dual CPU System

Vertical waste is reduced, as if a main memory access causes one thread to wait, the other thread can continue. Horizontal waste can also be reduced, as two discrete threads are vying for execution resources. While the downside is that contention will inevitably occur for execution resources, the overall effect should be that a physical CPU will be more productive. Multiple instruction execution units, e.g. integer and floating point, are a prerequisite for reasonable performance.

32.2.3 Use of Execution Resources

There are three categories of execution resource usage:

- **Replication.** These are controlling items where one copy is required for each logical CPU. Examples include: Instruction pointer; Register Renaming Logic, ITLB (Instruction Translation Lookaside Buffer)

- **Partitioning** allows certain resources to be split to handle multiple sets of instructions. Each logical CPU will have its own set of re-order buffers, load/store buffers and certain queues, e.g. a scheduling queue. Note that queues are split in one of two ways, statically or dynamically; statically is where one contiguous half of the queue is assigned to one logical CPU and the other half to the second logical CPU; whereas dynamic partitioning allows a more flexible use of the queue, the only constraint being that a logical CPU is not allowed to use the entire queue to the exclusion of the other logical CPU. The scheduling queues are dynamic. The reason is that the scheduler itself is not concerned with logical CPUs *per se*, it simply sees a continuous stream of instructions

- **Sharing.** Certain items are totally oblivious to the fact that SMT is being used (or not). Examples include the caches, instruction execution units (integer, floating point and load/store), and micro-architecture registers.

32.2.4 Modes of Operation

There are two modes of operation, MT and ST. MT, which stands for multitasking, is the hyper-threaded mode. However, there can be periods of time when there is insufficient work to warrant hyper-threading; say there is just a single process thread running. In this case the operating system can issue a HALT instruction which will cause the processor to switch to ST (Single Task) mode. ST mode allows the processor to use all resources, i.e. there are no hyper-threaded overheads and it can function as a standard processor. MT mode can be re-instigated when required by issuing another HALT instruction.

32.2.5 Operating System Considerations

Apart from the basic ability to recognise logical CPUs and thereby make use of hyper-threading the main consideration is a sensible distribution of work over the logical CPUs. To take a simple example, Figure 32-3 shows four logical CPUs spread over two physical CPUs. The first thread is dispatched to LCPU0. If the next thread was dispatched to the other logical CPU on the same physical processor the hyper-threading overheads that result from two threads sharing a physical processor are invoked immediately. The better performing approach is to dispatch it to the other physical processor (via LCPU1) so that the hyper-threading overheads are avoided for the moment. The third thread will be dispatched back on the first physical processor (via LCPU2) and the fourth thread on the second physical processor (via LCPU3). This mechanism allows work to be spread evenly over the available physical processors and thereby it typically provides better performance by minimising the overheads of processor sharing. Taking the Windows operating system as an example, Windows 2003 implements this preferred approach; Windows 2000 does not – hence the tales of performance degradation and the need to disable hyper-threading. Just for completeness on the Windows operating system front, NT does not support hyper-threading.

It should be recognised that dispatching work to CPUs is a more complex subject and there are other factors to take into account, e.g. the preference to dispatch a thread back onto the physical CPU where it last ran in the expectation that the processor cache may still contain relevant data and thereby cache misses will be minimised.

32.3 Performance

Intel claims up to a 30% performance improvement with hyper-threading. It is important to be clear what performance improvement means. The CPU is servicing two concurrent threads. Contention for resources will occur with the inevitable delays. To use a very crude analogy, a 1GHz CPU with a 30% improvement translates into 2 logical CPUs each running at 0.65 GHz. It follows that while throughput will increase, as overall more CPU power is available, the CPU element of response times will degrade due to the "slower" CPUs (it is assumed that the application is not making use of parallel processing, i.e. the normal case for most commercial applications). In reality, at low utilisations, say less than 50%, response times are unlikely to degrade as the amount of contention between threads will be negligible. There again, why use hyper-threading if not to eke out a fraction more CPU power by driving the system at higher utilisations?

On a general theme, the effectiveness of processor caching can have a significant effect on the performance of any CPU. There have always been question marks over the performance of systems that exhibit high memory volatility, particularly DBMSs. On a hyper-threaded system the caches are shared, leading to the likelihood of more cache misses and hence more main memory accesses. Some optimists claim that two threads sharing data in the cache can lead to improved performance, as fewer overheads are required for cache coherence across an SMP system. Well yes, if the assumption is true but it is frequently not the case in commercial applications. My current view is that I would tend not to recommend the use of hyper-threading for DBMS servers, or for application servers with large volatile memory heaps (say over 1GB).

Limited testing on one project produced the following results:

- Experimentation appears to justify Intel's claim that ST mode performance is equivalent to that of a non-hyper-threaded CPU

- Benchmark tests on a dual CPU system with 2.8 GHz Xeons running Microsoft Terminal Server with varying numbers of simulated users (up to 100) appeared to indicate that with hyper-threading enabled there was a capability to support more users than a non-hyper-threaded system; improvements were in the range of 20-30%. Unfortunately, as memory was the primary bottleneck to Terminal Server performance on this system, it effectively precluded high CPU utilisations and therefore the use of hyper-threading was ultimately somewhat academic.

A quick browse of the web indicates that many people are confused by the CPU utilisation metrics for hyper-threaded systems, and with good reason. I can find no authoritative explanations on how the metrics are collected. Indeed, I have contacted several individuals in this field but I have received no responses. My view is that the operating system will collect the time that each logical CPU is busy from the OS perspective. Each logical CPU can be 100% busy and therefore on a dual CPU system with 4 logical CPUs the maximum aggregate utilisation is 400%. As discussed, threads are competing for resources inside the physical CPU, contention that the operating system is totally unaware of. Using Intel's 30% improvement claim, a dual CPU system has a theoretical aggregate utilisation of 260% (1 * 1.3 * 2) but the metrics may show 400%. The difference (140%) is attributable to contention, i.e. it is not actual CPU usage *per se*. In short, the time that the operating system is capturing includes both usage and delays, and just to make life more complicated the delays will vary

depending on how busy the CPU actually is. A further twist is what happens when a CPU is running in Single Task mode for a spell; how does the operating system reflect the fact that it is running one CPU at full speed rather than two logical CPUs at reduced speeds in the metrics? These issues make it difficult to use detailed CPU metrics for sizing purposes with any degree of confidence. It would only seem appropriate to use the average CPU utilisation across the server simply as a relative metric for performance management purposes, e.g. 60% is satisfactory, 80% may require some remedial action. It is not valid to compare the total CPU utilisation of hyper-threaded and non-hyper-threaded systems; the former is based on logical CPUs and includes delays as discussed, while the latter is based on physical processors and does not include the same delays.

32.4 General Observations

The main observations are:

- Published benchmarks are limited to uniprocessor systems, which is not particularly helpful

- It is difficult to generalise on the throughput improvements that hyper-threading may bring, if any. It will vary from application to application; it is definitely an area where in-house benchmarking will be advantageous. As stated earlier, hyper-threading is not recommended for any system that exhibits high memory volatility, e.g. DBMS servers

- If sizing has to be done without the luxury of benchmarking it is suggested that the effect of hyper-threading is assumed to be broadly neutral, treating any potential gains as additional contingency rather than relying on any increased capacity

- The issues that surround the credibility of CPU utilisation figures make it advisable not to use metrics that have been captured from a hyper-threading system to size a non-hyper-threaded system, or vice versa

- As an aside, it is nothing to do with performance, beware of any software licensing implications when using hyper-threading. Many vendors use a per-processor method of charging and they may try to extend this to a per-logical CPU basis

- There is significant development in the areas of both SMT and CMP. IBM has introduced SMT on its Power 5 chips and Intel will extend it to Itanium chips, while larger CMP-based chips with more CPUs on a die are envisaged in the near future.

32.5 Further Reading

There is much material in this area. The following list is merely a sample.

Eggers, S., Levy, H., Simultaneous Multithreading Project at the University of Washington. The web site (www.cs.washington.edu/research/smt/) contains links to various useful articles in the area of SMT.

Intel Journal Volume 06 Issue 01 (February 2002), ISSN 1535-766X is dedicated to Hyper-threading. The complete issue can be downloaded in pdf format.

Dua, R.A, Lokhande, B., *A Comparative Study of SMT and CMP Multiprocessors* is a very useful paper.

Hyper-Threading Performance Analysis at 2cpu.com includes a number of benchmark figures.

There are a number of readable papers on the web that provide useful summaries of hyper-threading. Needless to say, they tend to cover similar ground. They include:

Hyper-Threading Technology on the Intel Xeon Processor Family for Servers is a white paper that is available on the Intel web site.

Stokes, J., *Introduction to Multithreading, Superthreading and Hyper-threading* can be found at arstechnica.com,

Vianney, D., *Hyperthreading speeds Linux* on the IBM developerWorks web site

Borozan, J., *Microsoft Windows-Based Servers and Intel Hyper-Threading Technology* can be found on the Microsoft web site.

