37 Simultaneous Multi-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.

This taster discusses the relatively recent introduction of SMT (Simultaneous Multithreading) techniques to increase throughput. It effectively replaces *Hyper-threading on SMP Systems* (issued in March 2005) which concentrated solely on Intel's hyper-threading implementation. This taster incorporates IBM's implementation on the Power5 chip and mentions future machines with more cores per chip and more threads per core.

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).*

37.1 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 a term used to describe the multi-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 the overhead of multi-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.

37.2 Technology Outline

37.2.1 Background

The *CPU Basics* taster describes the main techniques that are used to keep a modern 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 where multiple instruction execution units are available. Despite these optimisations on what are called superscalar CPUs, 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.

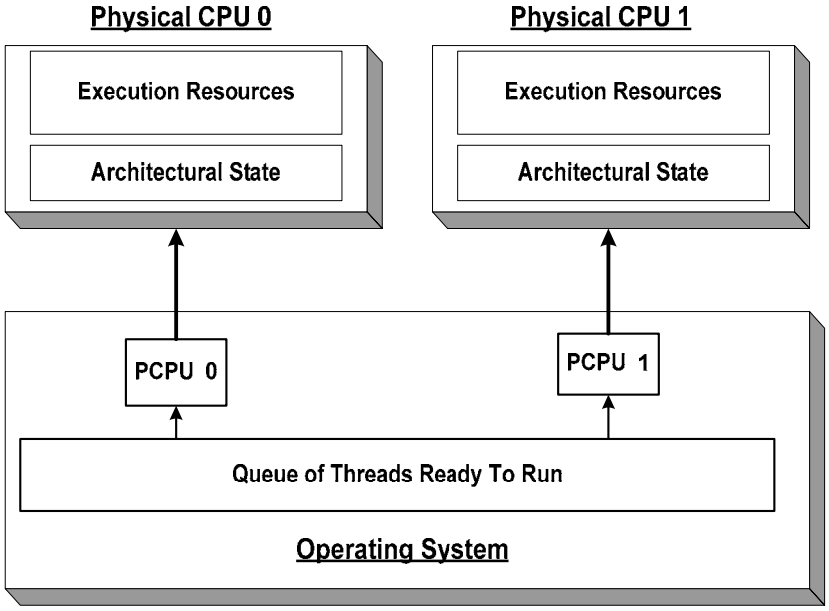

Figure 37-1 High Level View of Dual CPU System

The obvious route to improved throughput is to configure multiple CPUs, typically in an SMP system, as shown in Figure 37-1. See the *Multiprocessing (Shared Memory)* taster for further discussion of SMPs. 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 the ability to use CPU chips that contain two processor cores on a single die, sometimes called CMP (Chip Multiprocessors). As shown in Figure 37-2, 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 by all the cores. Note that some vendors, e.g. IBM include an off-chip level 3 cache in an attempt to further improve memory access performance.

The latest approach that has been brought to market is SMT (Simultaneous Multi-threading) where the objective is to make better use of each individual CPU by exploiting the superscalar features to allow thread level parallelism (TLP). Intel provided an early implementation on its Xeon and Pentium 4 processors under the soubriquet “hyper-threading”. IBM introduced SMT on its Power4 chips, refining the techniques further on the latest Power5 chips.

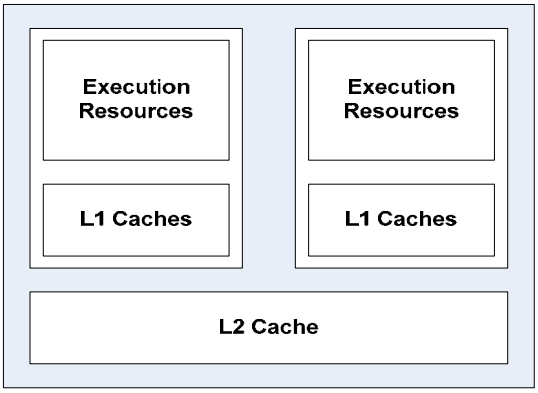

Figure 37-2 Dual Core Chip

37.2.2 SMT and The Logical CPU Concept

A conceptual diagram, as shown in Figure 37-3, shows that in SMT each physical CPU has multiple "architecture states", also called hardware contexts, limited to two in the diagram. Each architecture state needs its own set of registers. For example, the Intel implementation 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.

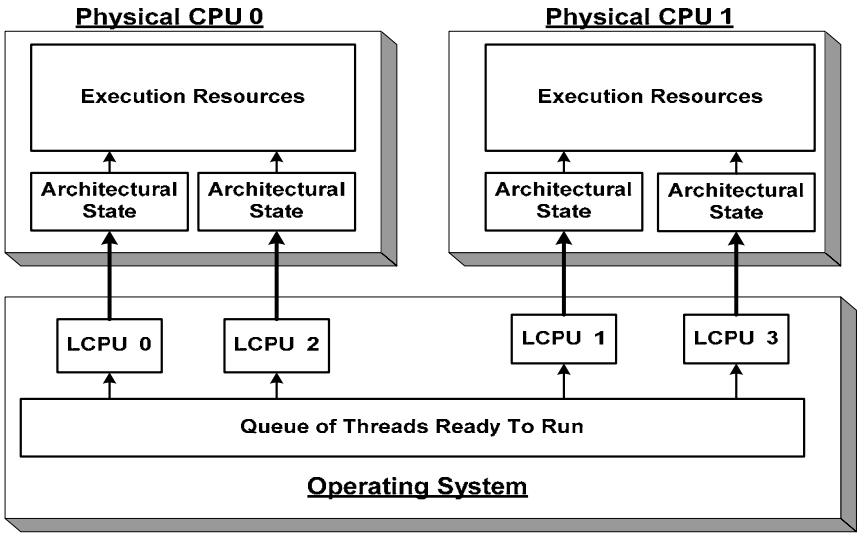

Figure 37-3 Overview of Multi-Threading on a Dual CPU System

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 use of the execution resources, rather than one. 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 per CPU, e.g. load, integer and floating point, are a prerequisite for reasonable performance.

37.2.3 Use of Execution Resources

There are three potential categories of execution resource usage on the chip. The observations below, while they are taken from the Intel implementation, are generally applicable.

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 may be 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. IBM's Power5 attempts to improve cache performance by increasing the set associativity of L1 and L2 caches (the greater the number of slots per set, the less likelihood there is of associativity misses). See the *Memory* taster for a description of set associativity.

Where resources are partitioned or shared, there ideally needs to be a mechanism which prevents one thread from monopolising the use of execution resources to the detriment of other threads. IBM's Power5 contains various facilities in the hardware, termed DRB (Dynamic Resource Balancing) to throttle back any offending thread.

37.2.4 Modes of Operation

From a hardware perspective there are two basic modes of operation, MT and ST. MT, which stands for multitasking, is the multi-threaded mode. However, there can be periods of time when there is insufficient work to warrant multi-threading; say there is just a single process thread running. In this case the operating system can issue a special instruction (HALT on Intel, mtctrl on IBM Power5) 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 multi-threaded overheads, and it can function as a standard processor. MT mode can be re-instigated when required by issuing another HALT or mtctrl instruction.

37.2.5 Operating System Considerations

There are three main issues to consider: thread scheduling, priorities and snoozing.

Scheduling. Apart from the basic ability to recognise logical CPUs and thereby make use of multi-threading the main consideration is a sensible distribution of work over the logical CPUs. To take a simple example, Figure 37-3 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 multi-threading overheads that result from two threads sharing a single physical processor are invoked immediately. The better performing approach is to dispatch the second thread to the other physical processor (via LCPU1) so that the multi-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 systems as an example, Windows 2003 implements this preferred approach; Windows 2000 does not – hence some of 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. IBM's Power5 uses the preferred scheduling approach.

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. it is preferable to dispatch a thread back onto the physical CPU where it last ran (not necessarily the same logical CPU) in the expectation that the processor cache may still contain relevant data and thereby any cache misses can be minimised.

Priorities. As mentioned under execution resource usage, there needs to be a mechanism which prevents one thread from monopolising the use of execution resources to the detriment of other threads. Ideally, the operating system, and to a lesser degree the application, should have the capability to set thread priorities. Obviously, this requires control of the whole stack (hardware, operating system and application) which may not be possible where commodity items are in use. IBM supports these priority features; Intel currently does not.

Snoozing. This is a feature which allows a short delay (ideally configurable) before the operating system decides to move from MT to ST mode, allowing any recently runnable task to execute quickly, and thereby avoiding the unnecessary overheads of two mode changes (to ST and back again to MT) .

37.2.6 Number of Threads Per Physical CPU

SMT implementations on current technology, general purpose superscalar chips produce optimum performance improvement with two threads (or logical CPUs) per physical CPU. It is questionable if any improvement can be achieved with 4 threads (or more). This is not to say that these higher thread levels are not possible, merely that it demands a different approach and the target market may be different, e.g. embedded systems, dedicated network servers, *et cetera* may be potential candidates. Sun has been promising a system code-named Niagara for some time. The initial implementations (2006?) are expected to be single CPU chip servers that contain 4, 6 or 8 cores, as opposed to the more typical dual cores of today, where each core can support up to 4 threads. Although the details are sketchy, it is likely that the individual cores will be thinner than the typical super-scalar core, in the sense that they will try to keep things simple. Rather than attempting to extract the last ounce of parallelism from a CPU, if a thread encounters a delay the objective will be to swap to another thread as quickly as possible. Such a system, possibly packaged in a blade server, is likely to be attractive to telecoms companies, where the emphasis is on maximising throughput even if single thread performance (and hence response times) may suffer. Intel's roadmap also includes large cores: Tanglewood is slanted to have 16 cores on a single chip.

One area of research is the possible use of "mini-threads" to support larger numbers of threads. The concept here is that an application that is running in a single hardware context (logical CPU) can create mini-threads, each with their own state, but sharing the hardware context.

37.3 Performance

Performance Improvements. There are various claims for the performance improvements that can result from the use of SMT. However, it is important to be clear precisely what "performance improvement" actually means. Let us say that the physical CPU is split into two logical CPUs and that it is servicing two concurrent threads, one per logical CPU. Contention for resources will occur with the inevitable delays. To use a very crude analogy, a 1GHz CPU with a 30% performance improvement translates into 2 logical CPUs each running at 0.65 GHz. It follows that while throughput will increase, as overall more CPU power can be consumed, 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 CPU utilisations, say less than 50%, response times are unlikely to degrade as the amount of contention between threads will be negligible. Degradation will occur at higher utilisations. There again, to realise the power and throughput advantages that SMT brings it will be essential to drive the system at higher utilisations! If it is not essential why use SMT?

Caching Issues. 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 an SMT 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. Ideally, the caches should be larger and/or improvements made to cache performance to offset any reduction in performance.

IBM Power5 Performance. Published results from IBM for a 4-way Power5-based system includes the following claims for performance improvements when using SMT: TPC-C - 30%; NetPerf - 50%; SPECjbb2000 - ~40%; and SPECsfs ~45%.

Intel Performance. In lieu of a reasonable set of published benchmarks, here are the results of limited testing on one project:

- Experimentation appeared 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.

Performance Monitoring. Accurate performance monitoring of SMT systems can be a serious problem, as the operating system needs to be aware of what is going on at the logical CPU level, which requires detailed communication between the hardware and the operating system. IBM's Power5 overcomes this issue by maintaining a "Processor Utilisation Resource Register" (PURR) per thread. AIX makes use of these registers to provide accurate process accounting and processor utilisation figures. However, at the other end of the spectrum, a quick browse of the web indicates that many people are confused by the CPU utilisation metrics for Intel-based 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.

37.4 General Observations

The main observations are:

- IBM's implementation of SMT is superior to the current Intel hyper-threading implementation on Xeons and Pentium4. This is due to various facts: IBM controls the whole stack, including hardware and operating system, whereas Intel are not in that fortunate similar situation; one of Intel's objectives was to minimise the increase in the size of the die to support SMT – IBM were not similarly constrained; SMT was only part of a series of changes that were incorporated into Power5 by IBM to increase performance, e.g. support for dual cores

- It is difficult to generalise on the throughput improvements that multi-threading may bring, if any. It will vary from application to application; it is definitely an area where in-house benchmarking will be advantageous

- If sizing has to be done without the luxury of benchmarking, particularly for Intel-based systems, it is suggested that the effect of SMT is assumed to be broadly neutral, treating any potential gains as additional contingency rather than relying on any theoretical gains in capacity

- The issues that surround the credibility of CPU utilisation figures on Intel systems 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 multi-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. Larger CMP-based chips with more CPUs on a die are envisaged in the near future with the possibility of more threads per CPU, particularly for specialised server usage where throughput is of primary importance.

37.5 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.

Gibbs, B., Berres, F., Castillo, L., et al, *Advanced POWER Virtualization Architecture and Performance Considerations*, IBM RedBook, SG24-5768-00

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.

