

Performance Workloads in a Hardware Multi Threaded Environment

Bret Olszewski

Octavian F. Herescu

IBM Corp.
Austin, TX

Abstract

This paper describes the benefits of hardware multithreading (HMT) on IBM's eserver Pseries systems on commercial workloads. The HMT mechanism takes advantage of today's considerable gap between the speed of the processor and memory to increase throughput by overlapping memory fetches with computation. We analyze the performance improvement by measuring 5 commercial benchmarks: SDET, Netperf, OLTP, Websphere and SFS, which were run using the same hardware configuration with and without HMT enabled. We also discuss the characterization of this technique compared to single-threaded processors and determine that by enabling HMT, the throughput improvement is in the range of 10-20%, confirmed by the CPI measurements. Using the hardware performance monitor we study many other hardware counters that factored into the improvement, including HMT specific ones.

1 Introduction

System design trends have in recent years concentrated on increasing instruction level parallelism (ILP). Highly superscalar designs, when coupled with aggressive memory prefetching mechanisms, have proved highly effective in integer and floating point applications. However, ILP gains in large footprint commercial codes have been virtually stalled.

According to Moore's Law, the processor's speed doubles every 18 months. That has held true for the last 36 years. Unfortunately, the speed of the memory system has not increased at the same rate. The main improvements of the memory components have been in density and manufacturability and not in performance. This has resulted in increased amount of memory per system at reduced cost. But each new generation of technology makes the latency to memory become a greater factor and that has led to increasingly complex system structures to maximize overall hardware performance.

One method to improve system performance is to overlap memory accesses with other instructions. This can and has been done in a number of mechanisms. The object of this paper is a mechanism called hardware multithreading (HMT).

2 Hardware Multithreading

HMT is a technique for tolerating memory latency by utilizing cycles in the CPU that normally would be stalled waiting on memory accesses [1]. On a system, such as the server Pseries 660 6M1, that contains relatively large on chip L1 caches and a large L2 cache, the ratio of loads and stores to memory accesses is reduced to less than one hundred to one. This number is still high because it means

that the processor spends an average of about 1.0 cycles of waiting on memory for each instruction running commercial workloads. *Figure 1* shows an estimate of the time the CPU stays stalled.

Considering that the system used in this paper uses an RS64-IV processor, which can execute up to four instructions per cycle, the time spent waiting for memory is far more than the time the instruction spends in the processor.

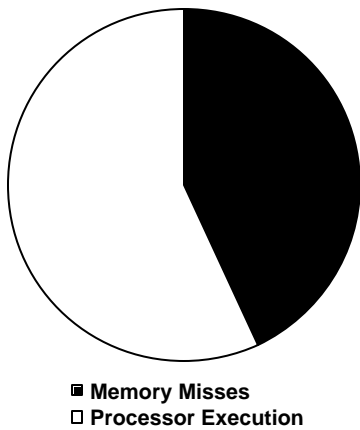


Figure 1 - CPI breakdown for uniprocessor

Hardware multithreading provides a mechanism for improving the overall system throughput by overlapping memory accesses with the execution of other instructions. This means that there will be multiple “active” threads per processor, which requires the replication of the processor-architected registers for each thread.

Cost savings, when compared to adding another physical processor, exist because replication is not required for the majority of the processor logic, such as: instruction cache, data cache, TLB, instruction fetch and dispatch mechanisms, branch units, fixed-

point units, floating-point units and storage-control units.

The sharing of so many units can take its toll on the performance of some workloads, one of which will be described in this paper. The RS64-IV processor has two contexts per physical processor. At any processor clock, only one of the logical processors (contexts) can have instructions in the execution pipeline. The other context is inactive. See *Figure 2*. The physical processor switches between the contexts at a fairly rapid rate giving the software the impression that it is dealing with two distinct processors. The fact that the processor used as an example in this paper has a four-stage pipeline allows a thread switch penalty of only three cycles.

2-way HMT

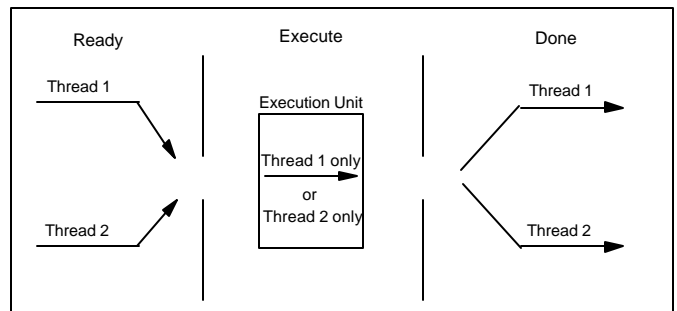


Figure 2 - HMT Only one thread active at any time

One of the requirements to make this method efficient and to achieve the improved performance is that the switch time between threads must be shorter than the latency of the event that triggered the switch. In our implementation the minimum penalty for an L1 miss is nine cycles. The thread switch penalty is only three cycles. *Figure 3* shows the switch timing for a L1 cache miss with an L2 cache hit. The thread switch penalty and the penalty for

TLB misses or instructions cache misses are comparable. As mentioned above, the processor switches between threads on selected events. The mechanisms to switch between contexts are controlled by the operating system and the events are selected by the thread switch control register (TSC). Three basic classes of switch events have been defined:

- ?? hardware events (cache misses and virtual memory translation misses).
- ?? software hints (software is allowed to have different priorities assigned to the logical processors).
- ?? time out (one logical processor cannot starve the other).

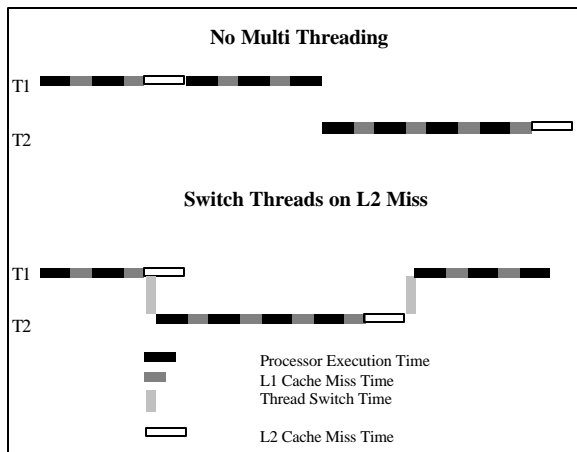


Figure 3 - Switching Penalty

The two logical processors may have different priorities. If the first thread on a processor is busy, the second will wait for an event that will trigger a logical processor switch. Hardware events are the main triggers of switching between the two logical processors. One of the most frequent causes for a switch is a cache miss. A cache miss occurs when

the active thread executes a load, store, or instruction fetch and the referenced data is not found in the cache. The cache miss causes the processor to switch to the other thread. The thread that was waiting on the secondary logical processor can now execute. But usually the execution is interrupted when the data needed by the first processor becomes available. The most unfortunate case is when the second thread also has to wait for data from memory. A switch can also occur when the hardware priority of the logical processors changes. We mentioned that each logical processor has its own hardware priority. Normally the hardware priority is 2 (medium priority) but software may switch the priority to a higher value (3 for example) using the *or r3,r3,r3 noop* or to a lower priority using the *or r1,r1,r1 noop* command. The hardware will switch immediately between threads when the priority is changed. The high priority **noop** can only be used in kernel mode.

To avoid starvation, one more event that can trigger a switch was introduced: time out. The hardware maintains a clock counting the number of cycles that the current thread has been executing. When the time out is exceeded, the hardware will switch to avoid thread starvation.

The HMT mechanism is particularly appropriate for the RS64 microprocessor. The logic overhead required to implement HMT is less than 10% of the chip size. The cost in cycle time to implement HMT is also small, less than 10%. A much more deeply pipelined processor could not so efficiently switch between threads due to its more complex state. On deeply pipelined processors, it is expected that simultaneous multi-threading (SMT), a mechanism

where multiple threads execute instructions at the same time, will prove the preferred solution. See *Figure 4*. SMT combines HMT with superscalar processor technology and allows multiple threads to issue instructions each cycle [3].

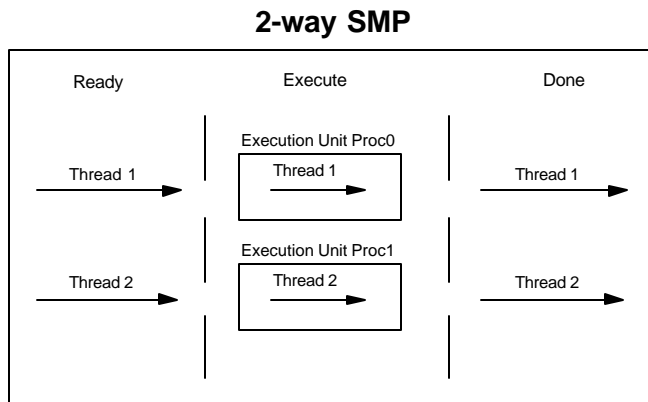


Figure 4 - SMT Two threads could be active at any time

3 AIX changes to implement HMT support

The first AIX version that supported HMT was 4.3.3. The HMT capabilities of AIX may be enabled or disabled by the *bosdebug* command followed by the *bosboot* command. The new mode will only be activated after a reboot. When HMT is fully enabled, the OS will see the system as having twice the number of processors; though the software which reports hardware configuration (e.g. *lscfg*) will report the number of physical processors. It is important to note that, unlike chips with independent cores [2], a hardware failure in the CPU core will disable both logical processors. Though the number of logical processors is doubled, each processor will typically have the performance of a little more than one half of the non-HMT processor.

There were very few changes required within AIX to support HMT. These changes were implemented

in system initialization, dispatching, idle process, and locking. The system initialization changes include modifications to enable the second logical processor, which required changes to start the secondary threads. Also, all the areas in the kernel that have to be aware of the difference between the real number of physical processors and the number of logical processors, were modified. For example, the operating system allocates only one set of mbuffs per physical processor saving memory.

By executing the command *netstat -m* the user will only see mbuffs for processors with even numbers: 0, 2, 4, etc. The operating system was modified to use software hints for low priority operations like the idle loop and lock spinning. Since HMT gives software the illusion of two logical processors per physical processor, each logical processor has operating system process management resources. This includes an idle thread per logical processor. The idle thread is optimized to run at a low hardware priority. This optimizes performance at low CPU utilization: if one logical processor is idle, the full resources of the physical processor are available to the other logical processor. Additionally, the areas associated with locking were modified. Every time the active thread acquires a kernel lock, the hardware thread priority is increased to high. When the lock is released, the priority is switched back to medium. This optimizes the cross section of critical code. Another performance improvement comes from the optimization to adjust hardware priority to low for code spinning on locks. This effectively yields the physical processor to the alternate thread. These two

changes tend to improve overall throughput because the cpu is kept busy during otherwise idle cycles.

The current hardware implementation restricts that external I/O interrupts are presented on only one logical processor per physical processor. This characteristic could affect some specific workloads, which will be detailed in the next chapters. The usage of thread priority software hints allows AIX to ensure that sufficient resources are available to service interrupts thus minimizing the shortcomings of the above asymmetric characteristics.

4 Performance Workloads

While quite a number of benchmarks have shown improvements when running over HMT enabled systems, in this paper we will present only a few representative workloads and describe the kind of benefits they get from HMT.

I. SPEC SDM SDET - Software Development Environment Throughput, is a benchmark that simulates a multiuser environment in which the users are executing randomly ordered scripts; basically, sets of AIX commands in shell scripts. One of the main reasons we chose this workload is that this benchmark stresses three main components of an Operating System: file management, processor management and the virtual memory manager. Over the years since SDET's first release, it has become clear that the benchmark's primary use has not been in evaluating how many work units a system performs per time, but how well a system responds under stress, and whether a set of operating system modifications will result in performance improvements.

II. Netperf - is a benchmark that can be used to measure various aspects of networking performance. Currently, its focus is on bulk data transfer (streaming) and request/response performance using either TCP, UDP, or the Berkeley Sockets interface. While this benchmark is now part of the public domain, IBM has developed a derivative tool which is more tightly integrated with the capabilities of the AIX operating system.

III. Online transaction processing workload (OLTP) simulates a complete computing environment, where a population of users executes transactions against a commercially available 64-bit database. The benchmark is centered around the principal activities (transactions) of an order-entry environment. This workload has the largest memory-footprint as well as executable size of the commonly executed performance workloads.

IV. Websphere eBusiness Benchmark - is an IBM internal benchmark used to evaluate performance using the IBM Websphere Application Server software product. The benchmark emulates the operation of an online brokerage firm. One or more network attached HTTP clients drive a server running servlets which use Enterprise Java Beans (EJBs) that service requests and access a relational database. The benchmark thus exercises the operating system using HTTP services, Java, and a relational database.

V. SPEC SFS97_R1 V3.0 - is a standard benchmark for evaluating the performance of a Network File System (NFS) file server. The variant used in this benchmark measures the NFS version 3 protocol over UDP. One or more clients drive a mix of

operations to a file server. The benchmark requires a large disk and network configuration to reach peak performance.

All of the above benchmarks were run on an RS/6000 model pSeries 660 6M1 with 8 processors running at 750 MHz, IBM SSA disks and varying amounts of RAM. The workloads are quite different in their consumption of CPU time resources. *Table 1* shows that the workloads are all CPU limited in the configurations used for this paper. The large amount of CPU time consumed in the operating system for netperf and sdet, which are used as operating system tests, is a marked difference from the more real-world OLTP and Websphere benchmarks. The NFS protocol is optimized to execute within the AIX kernel context, thus all execution time is system.

Workload	%User	%System	%I/O Wait	%Idle
Sdet	33	65	1	2
Netperf	1	99	0	0
OLTP	85	15	0	0
Websphere	66	33	0	0
SFS	0	98	2	0

Table 1 - non-HMT CPU Time

5 Hardware Measures

To facilitate an understanding of the benefits and effects of HMT on system performance, the aforementioned workloads were evaluated with HMT both enabled and disabled. The most important metric defining the benefit of HMT is the increase in workload throughput obtained while using it. *Chart 1* shows that the improvement observed is typically on the order of 10% to 20% for these workloads. The improvement in throughput is

directly related to the increase in instructions executed per unit time by the processors. This metric is typically defined as the cycles per instruction, or CPI.

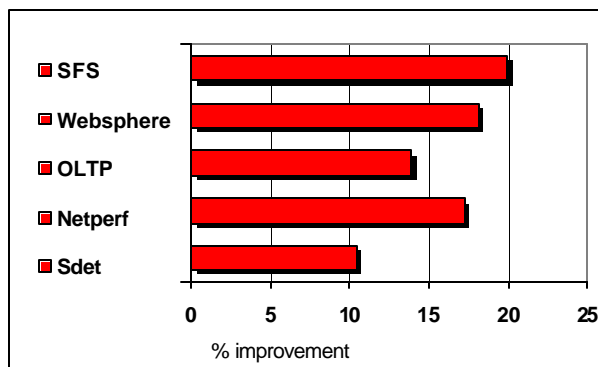


Chart 1 - Throughput Improvement with HMT

The hardware performance monitor can be used to assess the efficiency of the microprocessors with and without HMT enabled. There are a single set of hardware counters when HMT is enabled. The counters can be directed to count events for one or both hardware threads of execution for each physical processor, which allows us to directly count the total number of instructions executed by both hardware threads over a known number of cycles. The implementation of two threads sharing cache and translation resources would be expected to increase the frequency of cache misses. Since the L1 instruction and data caches as well as the TLB (translation look-a-side buffers) are relatively smaller than the L2 caches, they should see a larger increase in miss rates. As expected, the CPI with HMT enabled is lower, allowing higher throughput. This relative improvement in CPI is shown in *Chart 2*. Note that the improvements in CPI do not always scale linearly with throughput. This is due to

software efficiency as well as slight changes in the amount of idle time in the workload.

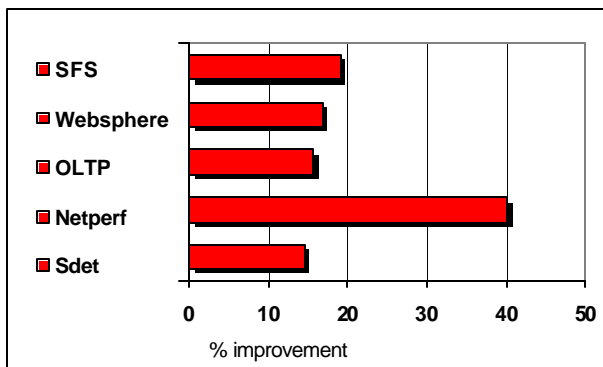


Chart 2 - CPI Improvement with HMT

Chart 3 shows the increase in instruction TLB (ITLB) miss rates with HMT. It would be expected that the largest increases would be in workloads with a multiple executables or very large executables.

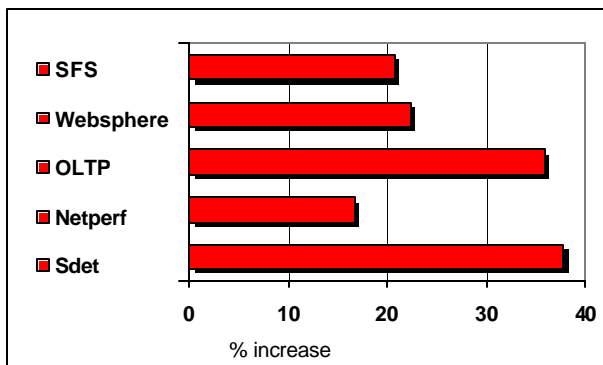


Chart 3 - Increase in ITLB per inst with HMT

This proves true for Sdet and OLTP, but the small increase in Websphere is surprising given its mix of large sized executables. As expected, netperf sees the smallest increase in ITLB miss rates, as the workload is dominated by kernel activity and a single small executable.

Chart 4 shows the increase in data translation lookaside (DTLB) misses per instruction with HMT. In this metric, the OLTP workload with its enormous data footprint and high multiprogramming level sees the greatest increase in DTLB misses.

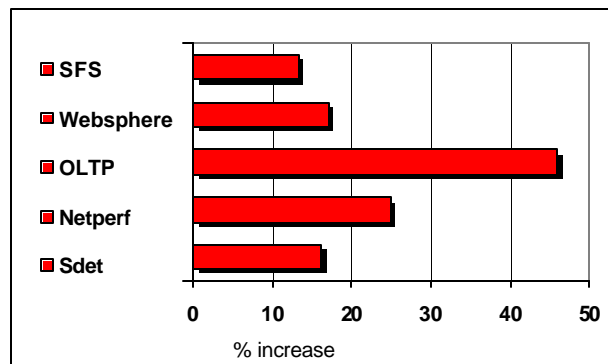


Chart 4 - Increase in DTLB per inst with HMT

The netperf workload, with a number of processes with small but essentially identical process address spaces, is penalized for aliasing of those addresses within the DTLB.

Chart 5 shows the increase in level-one instruction cache misses (IL1) with HMT. The increase in the IL1 cache miss rate for Netperf is somewhat misleading, as the absolute miss rates in each case are fairly low. This particular workload has a small instruction cache footprint. The increase in IL1 miss rate for Sdet is expected, as a large number of different executables are context switching within the shared IL1 cache with HMT. The increase observed in SFS has not been explained. Since the executable code is shared by all of the software threads, it is odd that instruction cache miss rate increases so greatly.

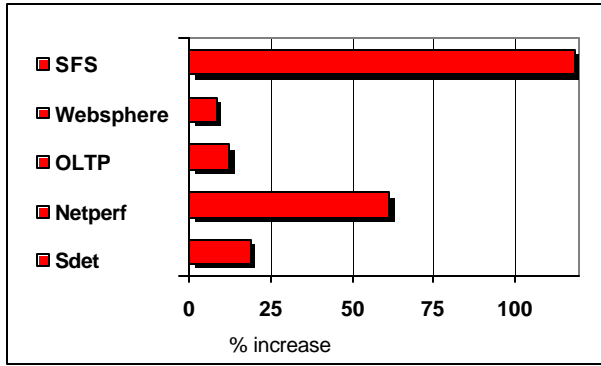


Chart 5 - Increase in IL1 per inst with HMT

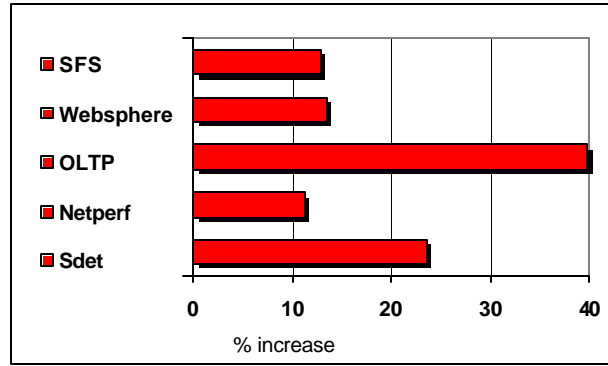


Chart 6 - Increase in DL1 per inst with HMT

Chart 6 shows the increase in level-one data cache misses (DL1) with HMT. Once again, the OLTP workload suffers the greatest increases with HMT for reasons cited with its DTLB miss rate. While the change in L1 miss rates are not typically crucial in performance for commercial workloads, an exception has been consistently observed running the SPEC CPU2000 SPECint_rate2000 workload. This workload runs multiple copies of the exact same programs in lockstep, which tends to result in extreme thrashing of the L1 data cache. On this workload, we consistently see a decrease in performance using HMT. While cache thrashing is fairly unlikely in complex environments, it may appear on systems with low multi-programming levels and long running jobs. The increase in miss rates for L1 caches and translation is indeed important, but the effect on the L2 is usually much greater due to memory latencies. Chart 7 shows in the studied benchmarks the increase is much lower on L2 miss rates due to the large size of the L2 caches on the system used. The decrease in L2 miss rate for Websphere is due to an actual decrease in the L2 misses resolved from another L2 cache.

This is actually a benefit of HMT sharing L2 cache between two logical processors. The reason we see a decrease in the L2 cache misses ratio for netperf is that while the actual number of misses increased, the ratio miss/instructions decreased because the machine has more idle cycles.

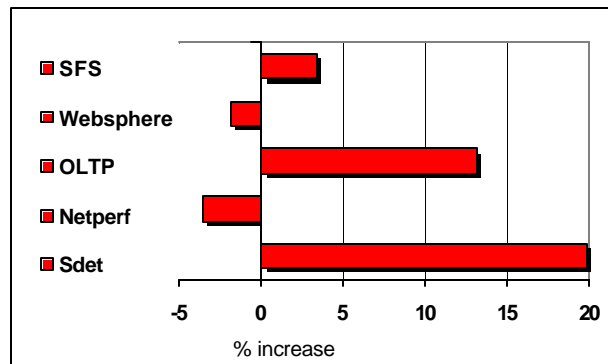


Chart 7 - Increase in L2 miss rates with HMT

We used the same workload for netperf as in the non-HMT case, but with HMT the system will be less utilized showing significant scaled throughput due to the increase in the number of CPUs. This somehow abnormal behavior is detailed in section 7. Though obvious, we must mention that adding HMT cannot improve performance if memory bandwidth

becomes constrained. Experiments on systems with bandwidth limitations have show very small gains in throughput. Another less studied issue is the effect on L2 miss rates with smaller L2 caches. There was concern that large footprint caches, such as OLTP, with very large instruction cache footprints, would scale poorly due to L2 cache thrashing. We were able to do some measurements on a one-processor 660 6F1 system using the OLTP workload. This system configuration has only a 2MB L2. The results of this measurement showed that the increase in L2 miss rates with a 2MB L2 were consistent with those observed on systems with 8MB L2 caches. A determination of miss rate increases below this size is difficult, as the 2MB L2 is still more than large enough to fully contain the instruction footprint. Since the instruction footprint is the most cacheable part of the workload, it seems likely the L2 miss rate increase will be larger with smaller caches.

6 HMT Specific Metrics

The hardware performance monitor also permits analysis of the HMT mechanism. These metrics provide a glimpse of how often there is switching between hardware threads as well as the causes of these switches. The rate of switching is a factor of cache and translation miss rates, software hints, and time-outs. *Chart 8* shows the average number of instructions between switches for the workloads. The hardware instrumentation also provides for counting the HMT thread switches, by the cause of the switch. *Charts 9,10,11,12, and 13* show the distribution of causes of most thread switch events for the three workloads.

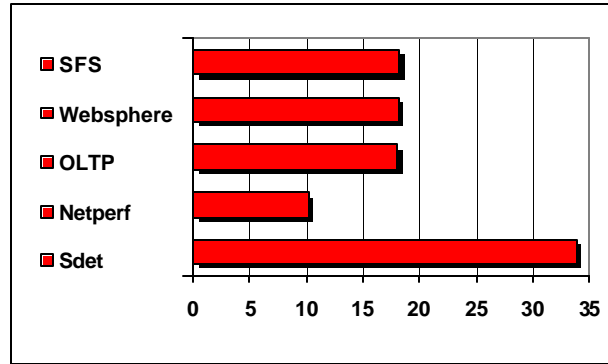


Chart 8 - Average instructions between switches

A few other thread switch cases, such as switch for L2 instruction miss, occur but are extremely infrequent.

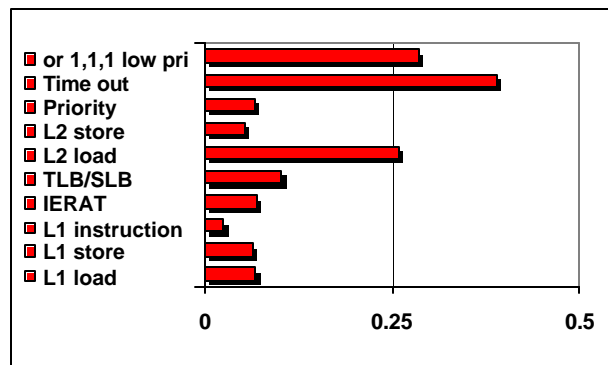


Chart 9 - Sdet HMT Fraction Switches

AIX configures the hardware to switch on all possible events. There are some circumstances when an L1 cache miss does not result in an immediate switch, which means the L2 switch events are not a subset of the L1 switch events.

A large fraction of switches are caused by the thread switch time-out. This time-out is set to a conservative value of 64 cycles. This value was selected to improve responsiveness for interrupt processing for high speed I/O adapters.

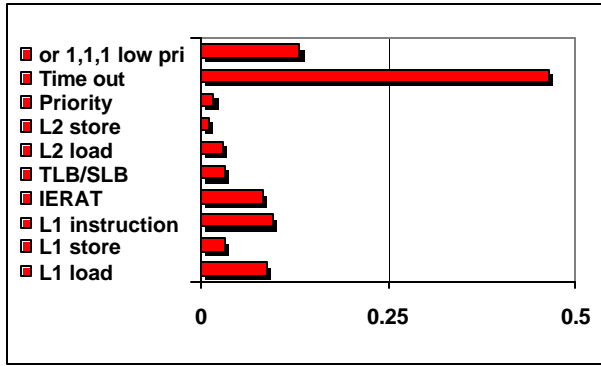


Chart 10 - Netperf HMT Fraction Switches

single threads that need to execute and complete in the shortest amount of time.

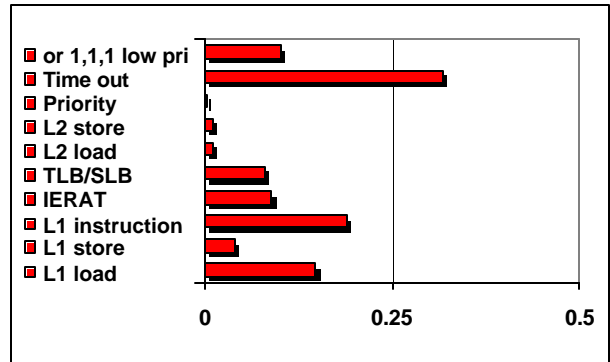


Chart 12 - Websphere HMT Fraction Switches

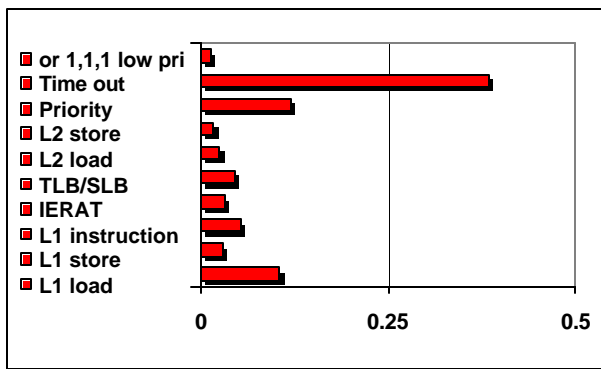


Chart 11 - OLTP HMT Fraction Switches

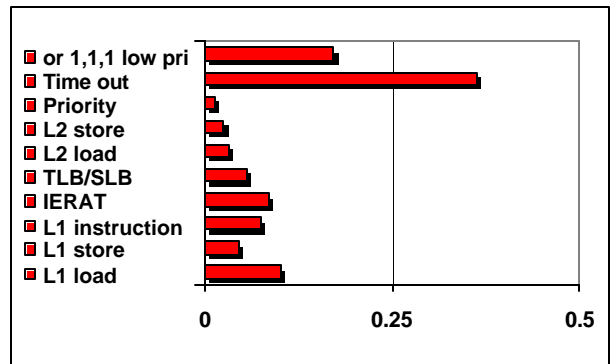


Chart 13 - SFS HMT Fraction Switches

A higher value of time-out would improve throughput, though at the cost of responsiveness. The *or 1,1,1 noop* instruction switches are typically the result of the idle process or lock spin switches for low priority.

7 Performance Tradeoffs

The nature of HMT performance enhancements assumes highly multi-programming workloads (meaning when a task is waiting another one is running). AIX's implementation of HMT attempts to provide fairness between the threads that are executing. While this balance is very effective for transactional environments, it is not optimal for

No circumstances will allow a thread to have a shorter execution time in an HMT enabled environment. Let's assume that there is only one thread that executes on a system that supports HMT. Whenever there is a cache miss, the system will switch to the other logical processor until the data for the miss becomes available. That means that there are two switches and each introduces a 3-4 processor cycles penalty. Had the system not switched, the running thread would not have been penalized. We saw the penalty in the time, using a business intelligence workload which measured the

run time per query. While the throughput increased when the queries were run in parallel, the time to run individual queries in isolation increased in a range from 2% to 20%. Since all the benchmarks we discuss in this paper ran for a predefined amount of time, this shortcoming of HMT could not be quantified. It is important to understand the type of workload that a system is most likely to handle and based on that, make a decision whether it is beneficial to enable HMT on the system. Thus, the benefits of HMT for highly transactional, highly multi-programming works must be weighted against the drawbacks for time-critical tasks.

Another side effect of HMT is a consequence of sharing the physical resources of one processor between the two logical processors. And that is an increase in variability in elapsed and CPU time to complete a task. AIX treats the two logical processors as equals and that means that the amount of resources available to one processor can, and is, affected by what is running on the other one. When HMT is fully enabled AIX behaves as if the system has doubled its number of processors. This increase could create scalability problems. Resource sharing also poses a challenge to the CPU monitorization tools. AIX determines the CPU utilization using a common sampling technique querying the tasks running on each processor 100 times per second. This implementation has each processor taking clock interrupts on average each 10 milliseconds. The hardware supports two mechanisms, one allowing the two logical processors to take clock interrupts in absolute time, the other to take them proportional to their execution time. Clearly, having 100 clock

interrupts per second delivered proportionally to the two logical processors would improve the accuracy of CPU utilization, since the logical processors may proceed at different rates. But many tools assume, based on constants in header files, absolute clock tick counts. Thus AIX maintains its original behavior of each processor taking 100 clock interrupts per second. Note here that this problem will need to be resolved if, as in the mainframe world, multiple operating instances coexist on a physical processor (e.g. shared processor partitions). The problem with constant rate sampling is the capacity implied by the CPU utilization. In essence, if the system is 50% busy and performs a certain number of tasks per second, one may assume that approximately twice as many tasks per second can run on the system before the CPU is totally consumed. The above assumption does not take into consideration the other factors in scaling a workload: memory, I/O bandwidth, or workload scalability. When HMT is enabled on the system, with the sampling technique, the Operating System will sample both logical processors every 1/100th of a second. If only one thread is running on the system at a certain time, and it blocks for a cache miss, the system will switch to the other processor, which will be idle since there is only one thread executing in the system. The operating system will record the elapsed time on this processor as idle time. If another thread is running on the system, does the throughput increase linearly with the idle time that was previously observed? The answer is: not necessarily, since both threads could end up blocked on cache misses from time to time.

The way Netperf reports the system utilization is by averaging the CPU utilization of all the CPU's that the system detects. With HMT enabled, there will be twice as many processors available. When using Netperf, we used the same workload on both cases HMT and nonHMT. In the nonHMT case the workload used would make the CPU utilization be 100%. Using the same workload with HMT the CPU monitoring tool would report the CPU's as only 93% busy in average. There is no guarantee that even if there were no other constraints, like memory or I/O bandwidth, the system would be able to do 7% more work, which implies the CPU utilization observed with HMT is somewhat less precise as a measure of available capacity. So, if a certain workload requires a highly accurate CPU utilization, HMT is not recommended. In the case of Netperf, measurements showed that even if the workload was increased the throughput did not increase. In fact the throughput decreased while the CPU reached 100%.

8 Conclusions

Our analysis has shown performance gains of ten to twenty percent, on different types of workloads with high multi-programming levels are possible. It is important to understand that the gains are very much dependent on the type of the user's workload. Issues with the performance of long running critical tasks, workload scalability, cache thrashing and CPU utilization should be taken in consideration and quantified in order to determine the applicability of HMT in a specific user environment.

For future work we think that improvements could be done in both hardware and software by keeping these two goals in mind:

- 1) We need to have hardware that has a low switching cost, since that is the major overhead, and
- 2) To be able to provide good single-thread performance, therefore allowing applications with low parallelism to execute efficiently.

References

- [1] J. Borkenhagen, R. Eickemeyer, and R. Kalla :A Multithreaded PowerPC Processor for Commercial Servers, IBM Journal of Research and Development, November 2000, Vol. 44, No. 6, pp. 885-98.
- [2] Gulati, M., Bagherzadeh, N.: Performance Study of a Multithreaded Superscalar Microprocessor. 2nd Int. Symp. On High Performance Computer Architectures, February 1996, 291-301
- [3] Susan Eggers, Joel Emer, Henry Levy, Jack Lo, Rebecca Stamm, and Dean Tullsen, *IEEE Micro*, September / October 1997, pages 12-18

Acknowledgements

Jim Van Fleet, Mysore Srinivas and Greg Mewhinney for their earlier work on HMT on AIX.

Steve Kunkel for RS64 hardware help.

Augie Mena, Sujatha Kashyap, Qunying Gao and Anthony Garcia for helping us collect the data.

Trademarks

SPEC, SPEC SFS97_R1, SPEC SDM SDET, SPEC CPU2000 SPECint_rate2000 are trademarks of Standard Performance Evaluation Corporation.

IBM, AIX, RS/6000, and Websphere are a trademark of International Business Machines.