Evaluating the Impact of Simultaneous Multithreading on Network Servers Using Real Hardware

Yaoping Ruan yruan@cs.princeton.edu
Vivek S. Pai vivek@cs.princeton.edu
Erich Nahum† nahum@watson.ibm.com
John M. Tracey† traceyj@us.ibm.com

Department of Computer Science, Princeton University, Princeton, NJ 08544
† IBM T.J.Watson Research Center, Yorktown Heights, NY 10598

ABSTRACT

This paper examines the performance of simultaneous multithreading (SMT) for network servers using actual hardware, multiple network server applications, and several workloads. Using three versions of the Intel Xeon processor with Hyper-Threading, we perform macroscopic analysis as well as microarchitectural measurements to understand the origins of the performance bottlenecks for SMT processors in these environments. The results of our evaluation suggest that the current SMT support in the Xeon is application and workload sensitive, and may not yield significant benefits for network servers.

In general, we find that enabling SMT on real hardware usually produces only slight performance gains, and can sometimes lead to performance loss. In the uniprocessor case, previous studies appear to have neglected the OS overhead in switching from a uniprocessor kernel to an SMT-enabled kernel. The performance loss associated with such support is comparable to the gains provided by SMT. In the 2-way multiprocessor case, the higher number of memory references from SMT often causes the memory system to become the bottleneck, offsetting any processor utilization gains. This effect is compounded by the growing gap between processor speeds and memory latency. In trying to understand the large gains shown by simulation studies, we find that while the general trends for microarchitectural behavior agree with real hardware, differences in sizing assumptions and performance models yield much more optimistic benefits for SMT than we observe.

Categories and Subject Descriptors: C.4 PERFORMANCE OF SYSTEMS:Design studies

General Terms: Measurement, Performance.

Keywords: Network Server, Simultaneous Multithreading(SMT).

1. INTRODUCTION

Simultaneous multithreading (SMT) has recently moved from simulation-based research to reality with the advent of commercially available SMT-capable microprocessors. Simultaneous multithreading allows processors to handle multiple instruction streams in the pipeline at the same time, allowing higher functional unit utilization than is possible from a single stream. Since the hardware support for this extra parallelism seems to be minimal, SMT has the potential to increase system throughput without significantly affecting system cost. While academic research on SMT processors has been taking place since the mid-1990's [8, 37], the recent availability of SMT-capable Intel Xeon processors allows performance analysts to perform direct measurements of SMT benefits under a wide range of workloads.

One of the biggest opportunities for SMT is in network servers, such as Web, FTP, or file servers, where tasks are naturally parallel, and where high throughput is important. While much of the academic focus on SMT has been on scientific or computation-intensive workloads, suitable for the High Performance Computing (HPC) community, a few simulation studies have explicitly examined Web server performance [18, 26]. The difficulty of simulating server workloads versus HPC workloads is in accurately handling operating system (OS) behavior, including device drivers and hardware-generated interrupts. While processor-evaluation workloads like SPEC CPU [33] explicitly attempt to avoid much OS interaction, server workloads, like SPECweb [34] often include much OS, filesystem, and network activity..

While simulations clearly provide more flexibility than actual hardware, evaluation on real hardware also has its advantages, including more realism and faster evaluation. Using actual hardware, researchers can run a wider range of workloads (e.g., bottom-half heavy workloads) than is feasible in simulation-based environments. Particularly for workloads with large data set sizes that are slow to reach steady state, the time difference between simulation and evaluation can be substantial. The drawback of hardware, however, is the lack of configuration options that is available in simulation. Some flexibility in the hardware analysis can be gained by using processors with different characteristics, though this approach is clearly much more constrained than simulators.

This paper makes four contributions:

- We provide a thorough experimental evaluation of SMT for network servers, using five different software packages and three hardware platforms. We believe this study is more complete than any related work previously published.
- We show that SMT has a smaller performance benefit than expected for network servers, both in the uniprocessor and dual-processor cases. In each case, we identify the macro-level issues that affect performance.
- We perform a microarchitectural evaluation of performance using the Xeon's hardware performance counters. The results provide insight into the instruction-level issues that affect performance on these platforms.
- We compare our measurements with earlier simulation results to understand what aspects of the simulated processors

SIGMETRICS'05, June 6–10, 2005, Banff, Alberta, Canada.

yielded much larger performance gains. We discuss the feasibility of these simulation models, both in the context of current hardware, and with respect to expected future trends.

Our evaluation suggests that the current SMT support is sensitive to application and workloads, and may not yield significant benefits for network servers, especially for OS-heavy workloads. We find that enabling SMT usually produces only slight performance gains, and can sometimes lead to performance loss. In the uniprocessor case, simulations appear to have neglected the OS overhead in switching from a uniprocessor kernel to an SMT-enabled kernel. The performance loss associated with such support is comparable to the gains provided by SMT. In the 2-way multiprocessor case, the higher number of memory references from SMT often causes the memory system to become the bottleneck, offsetting any processor utilization gains. This effect is compounded by the growing gap between processor speeds and memory latency. We find that SMT on the Xeon tends to provide better gains when coupled with large L3 caches. By comparing performance gains across variants of the Xeon, we argue that such caches will only become more crucial for SMT as clock rates increase. If these caches continue to be one of the differentiating factors between commodity and higher-cost processors, then commodity SMT will see eroding gains going forward. We believe this observation also applies to architectures other than the Xeon, since SMT only yields benefits when it is able to utilize more processor resources.

Using these results, we can also examine how simulation suggested a much more optimistic scenario for SMT, and why it differs from what we observe. For example, when calculating speedups, none of the simulations used a uniprocessor kernel when measuring the non-SMT base case. Furthermore, the simulations use cache sizes that are larger than anything commonly available today. These large caches appear to have supported the higher number of threads used, yielding much higher benefits than what we have seen, even when comparing with the same number of threads. We do not believe that the processor models used in the simulation are simply more aggressive than what is available today or likely to be available in the near-future. Instead, using comparable measurements from the simulations and existing hardware, we show that the type of processors commonly modeled in the simulations are unlikely to ever appear as slightly-modified mainstream processors. We argue that they have characteristics that suggest they could be built specifically for SMT, and would sacrifice single-thread performance.

The rest of this paper is organized as follows: we provide some background on SMT, the Xeon, and our experimental setup in Section 2. We measure SMT's effect on throughput and perform a microarchitectural analysis in Sections 3 and 4. In Section 5 we compare our measurement results to previous simulation studies. The impact of other workloads is discussed in Section 6. Section 7 discusses related work, and we conclude in Section 8.

2. BACKGROUND

In this section we present an overview of the Intel Xeon processor with Hyper-Threading (Intel's term for SMT), then describe our experimental platform including hardware parameters and server configuration, our workloads and measurement methodology.

2.1 SMT Architecture

The SMT architecture was proposed in the mid-1990's, and has been an active area for academic research since that time [16, 36, 37], but the first general-purpose processor with SMT features was not shipped until 2003. The main intent of SMT is to convert thread-level parallelism into instruction-level parallelism. In SMT-enabled processors, instructions from multiple processes or threads can be fetched together, without context switching, and can be executed simultaneously on shared execution resources. From either the operating system's or user program's perspective, the system appears to have multiple processors. Currently, we are aware of only two processors in production that support SMT – the Intel Xeon with Hyper-Threading and the IBM POWER5. The Xeon has been available longer, and since it is available in a wide range of configurations, it provides us with an opportunity to affordably evaluate the impact of several features.

clock rate	2.0 or 3.06 GHz
pipeline	20 stages or 30 stages starting from TC
Fetch Policy	6 μops per cycle round robin for logical processors
Retirement	3 μops per cycle
Shared Resources	caches, branch predictors, decoder logic DTLB, execution units, buses
Duplicated Resources	interrupt controller, status registers ITLB, renaming logic
Partitioned Resources	μop queue, re-ordering buffer load/store buffer, general instruction buffer

Table 1: Intel Xeon hardware parameters.

The Xeon is Intel's server-class x86 processor, designed to be used in higher-end applications. It is differentiated from the Pentium 4 by the addition of extra on-chip cache, support for SMT (though this is now beginning to appear on standard P4 processors), and on-chip support for multiprocessing. It is a superscalar, out-of-order processor with a deep pipeline, ranging from 20 to 30 stages depending on processor version and clock speed. It has two hardware contexts (threads) per processor, which share most of the resources, such as caches, execution units, branch predictor, control logic, and buses. Its native x86 instruction set architecture is CISC, but it internally translates instructions into RISC-like micro-operations (μops) before executing them. Buffering queues between major pipeline logic blocks, such as μop queues, and the reorder buffer, are partitioned when SMT is enabled, but are recombined when only one software thread is active [17]. The basic hardware information for the Xeon can be found in Table 1.

2.2 Experimental setup

To reduce the number of variables in our experiments, all of our tests use the same motherboard, an Intel SE7505VB2 with 4GB memory, which is capable of supporting up to two processors. Our processors are the 3.06 GHz Xeon with no L3 cache, the 3.06 GHz Xeon with a 1MB L3 cache, and the 2.0 GHz Xeon without L3 cache. Using these three processors, we can determine the effect of different clock rates, and the effect of the presence or absence of an L3 cache. All processors have a 533 MHz front-side bus (FSB). The 2.0 GHz use a 20-stage pipeline starting from the trace cache (TC), while the 3.06 GHz Xeons use a 30-stage pipeline. All tests use the same physical motherboard, and we manually replace processors as needed, in order to reduce the chance that variations in memory manufacturing, etc., can affect the results. The memory hierarchy details for our system are provided in Table 2. Using lmbench [19], we find the main memory latencies are 225 cycles for the 2.0 GHz Xeon, 320 cycles for the 3.06 GHz Xeon with L3, and 344 cycles for the 3.06 GHz processor without L3 cache.

The increase in memory latency (measured in cycles) for the 3.06 GHz processors is not surprising, since the cycles are shorter in

Level	Capacity	Associativity	Line Size	Latency (cycles)
TC	12K μops	8 way	6 μops	N/A
D-L1	8 KB	4 way	64 bytes	2
L2	512 KB	8 way	128 bytes	18
Memory	4 GB	N/A	N/A	225 - 344
ITLB	128 entries, 20 cycles miss penalty			
DTLB	64 entries, 20 cycles miss penalty			

Table 2: Intel Xeon memory hierarchy information. The latency cycles of each level of the memory hierarchy includes the cache miss time of the previous level

absolute time. The absolute latency is relatively constant since the FSB speed is the same. The impact on bandwidth is 22%, much less than the clock speed difference – the 2.0 GHz system has a read bandwidth of 1.8 GB/sec while the 3.06 GHz system has a value of 2.2 GB/sec. While higher bandwidth is useful for copy-intensive applications, the memory latency is more important to applications that perform heavy pointer-chasing. Early Web servers performed significant numbers of memory copies to transfer data, but with the introduction of zero-copy [22] support into servers, copy bandwidth is less of an issue.

Our testing harness consists of 12 uniprocessor client machines with AMD Duron processors at 1.6 GHz. The aggregate processor power of the clients are enough to ensure that the clients are never the bottleneck. To ensure adequate network bandwidth, the clients are partitioned into four groups of three machines. Each group is connected to the server via a separate switched Gigabit Ethernet, using four Intel e1000 MT server adapters at the server.

We compare five different OS/processor configurations, based on whether a uniprocessor or multiprocessor kernel is used, and whether SMT is enabled or disabled. Using the BIOS support and OS boot parameters, we can select between one or two processors, and enable or disable SMT. For most of our tests, we use a multiprocessor-enabled (SMP) kernel, since the OS sees an SMT-enabled processor as two logical processors. However, when we run with one physical processor and SMT disabled, we also test on a uniprocessor kernel. These combinations yield the five configurations studied in this paper: one processor with uniprocessor kernel (1T-UP), one processor with SMP kernel (1T-SMP), one processor with SMP kernel and SMT enabled (2T), two processors (2P), and two processors with SMT enabled (4T). Key features of the five configuration and their names used in this paper are shown in Table 3. The operating system on the server is Linux, with kernel version 2.6.8.1. This version includes optimizations for SMT, which we enable. The optimizations are described next.

2.3 Kernel Versions and Overheads

In evaluating SMT performance on uniprocessors, it is important to understand the distinction between the types of kernels available, because they affect the delivered performance. Uniprocessor kernels, as the name implies, are configured to only support one processor, regardless of how many physical processors are in the system. Multiprocessor kernels are configured to take advantage of all processors in the system using a single binary image. While intended for multiple processors, they are designed to operate without problems on a single processor.

Uniprocessor kernels can make assumptions about what is possible during execution, since all sources of activity are taking place on one processor. Specifically, the OS can make two important assumptions: that only one process or thread can be actively running in the kernel at once, and that when the kernel is executing on behalf of that process or thread, the only other source of execution is hardware interrupts. The first condition is important for protecting data in the kernel – when the kernel is executing, it generally does not have to worry about locking kernel data structures unless it may block on some resource. The only data sharing that remains is for data used by any interrupt servicing code. The existence of only one processor also simplifies this code, since it can simply disable interrupts when manipulating such data, and enable interrupts after the critical section. Since enabling or disabling interrupts is a single instruction on the x86, this code can be compact.

	1T-UP	1T-SMP	2T	2P	4T
# CPUs	1	1	1	2	2
SMP kernel	No	Yes	Yes	Yes	Yes
SMT enabled	No	No	Yes	No	Yes

Table 3: Notation used in this paper reflecting different hardware and kernel configurations

On multiprocessors (SMP), the invalidation of both assumptions causes the need to have more synchronization code in the kernel, leading to more overhead. Both processors can be executing kernel code simultaneously, so any global data in the kernel must be protected from race conditions. The simplest approach, using a "giant kernel lock" to ensure only one processor is in the kernel at a time, reduces the performance of OS-intensive workloads, and has been replaced with fine-grained locking on all major OSes. Interrupt handling must also differ – since interrupts can be delivered to a different processor than the one using data shared with the interrupt handler, the kernel cannot simply locally disable interrupts. Instead, all data accessible by an interrupt handler must also be protected using locks, to prevent another processor from accessing it simultaneously.

For uniprocessors, running a multiprocessor version of the kernel can therefore cause a much larger performance loss than might be expected, because instead of one extra lock operation per system call, many lock operations may be necessary for fine-grained data sharing. For network servers, this overhead can be significant, if every packet and acknowledgment invokes extra code that is not necessary in the uniprocessor case.

Since the OS treats an SMT-enabled processor as two logical processors, it must use the SMP kernel, with the associated overheads. Kernel designers have taken steps to reduce some overheads, knowing that some operations can be performed more efficiently on an SMT with two logical processors than a multiprocessor with two physical processors. However, since SMTs interleave instructions from multiple contexts, these overheads cannot be reduced to the level of uniprocessor kernels. The Linux kernel implements a number of SMT-specific optimizations, mostly related to processor affinity and load balancing [3]. Task run queues are shared between contexts on each physical processor, eliminating the chance of one context being idle while the other has multiple tasks waiting. This balancing occurs whenever a task wakes up or when any other task on the same physical processor finishes. Processor affinity, intended to minimize cache disruption, is also performed on physical processor instead of logical processors.

2.4 Test & Measurement methodology

We focus on Web (HTTP) servers and workloads because of their popularity and the diversity of server implementations available. The server applications we use are Apache 2.0 [2], Flash [21],

TUX [38], and Haboob [40]. Each server has one or more distinguishing features which increases the range of systems we study. All of the servers are written in C, except Haboob, which uses Java. TUX is in-kernel, while all of the others are user-space. Flash and Haboob are event-driven, but Haboob also uses threads to isolate different steps of request processing. We run Apache in two configurations – with multiple-processes (dubbed Apache-MP), and multiple threads (dubbed Apache-MT) using Linux kernel threads, because the Linux 2.6 kernel has better support for threads than the 2.4 series, and the Xeon has different cache sharing for threaded applications. Threaded applications share the same address space register while multi-process applications usually have different registers. Flash has a main process handling most of the work with helpers for disk IO access. We run the same number of Flash main processes as the number of hardware contexts. TUX uses a thread-pool model, where multiple threads handle ready events. With the exception of Haboob, all of the servers use the zero-copy interfaces available on Linux, reducing memory copy overhead when sending large files. For all of the servers, we take steps described in the literature to optimize their performance. While performance comparison among the servers is not the focus of this paper, we are interested in examining performance characteristics of SMT on these different software styles.

We use the SPECweb96 [34] benchmark mostly because it was used in previous simulation studies. Compared to its successor, the SPECweb99 benchmark, it spends more time in the kernel because all requests are static, which resembles other server workloads such as FTP and file servers. We also include SPECweb99 benchmark results for comparison. SPECweb is intended to measure a self-scaling capacity metric, which means that the workload characteristics change in several dimensions for different load levels.

To simplify this benchmark while retaining many of its desirable properties, we use a more tractable subset when measuring bandwidths. In particular, we fix the data set size of the workload to 500MB, which fits in the physical memory of our machine. We perform measurements only after an initial warm-up phase, to ensure that all necessary files have been loaded into memory. During the bandwidth tests, no disk activity is expected to occur. We disable logging, which causes significant performance losses in some servers. SPECweb99 measures the number of simultaneous connections each server is able to sustain while providing the specified quality of service to each connection. The SPECweb99 client software introduces latency between requests to decrease the per-connection bandwidth. SPECweb96 does not have this latency, allowing all clients to issue requests in a closed loop, infinite-demand model. We use 1024 simultaneous connections, and report the aggregate response bandwidth received by the clients.

We use a modified version of OProfile [20] to measure the utilization of microarchitectural resources via the Xeon's performance-monitoring events. OProfile ships with the Linux kernel and is able to report user, kernel or aggregated event values. OProfile operates similarly to DCPI [1], using interrupt-based statistical sampling of event counters to determine processor activity without much overhead. We find that for our experiments, the measurement overhead is generally less than 1%. While OProfile supports many event counts available on the Xeon, we enhance the released code to support several new events, such as L1 data cache miss, DTLB miss, memory loads, memory stores, resource stalls, etc.

3. SMT PERFORMANCE

In this section we evaluate the throughput improvement of SMT in both uniprocessor and multiprocessor systems. Particular attention is given to the comparison between configurations with and without SMT enabled, and kernels with and without multiprocessor support. We first analyze trends at a macroscopic level, and then use microarchitectural information to understand what is causing the macroscopic behavior. Our bandwidth result for the basic 3.06 GHz Xeon, showing five servers and five OS/processor configurations, can be seen in Figure 2. Results for 2.0 GHz and 3.06 GHz with L3 cache are seen in Figures 1 and 3, respectively. For each server, the five bars indicate the maximum throughput achieved using the specified number of processors and OS configuration.

While bandwidth is influenced by both the server software as well as the OS/processor configuration, the server software usually has a large effect (and in this case, dominant effect) on bandwidth. Heavily-optimized servers like Flash and TUX are expected to outperform Apache, which is designed for flexibility and portability instead of raw performance. The relative performance of Apache, Flash, and Haboob is in-line with previous studies [28]. TUX's relative performance is somewhat surprising, since we assumed an in-kernel server would beat all other options. To ensure it was being run correctly, we consulted with its author to ensure that it was properly configured for maximum performance. We surmise that its performance is due to its emphasis on dynamic content, which is not exercised in this portion of our testing. Haboob's low performance can be attributed both to its use of Java as well as its lack of support for Linux's sendfile system call (and as a result, TCP checksum offload). For in-memory workloads, the CPU is at full utilization, so the extra copying, checksumming, and language-related overheads consume processor cycles that could otherwise be spent processing other requests.

3.1 SMP Overhead on Uniprocessor

We can quantify the overhead of supporting an SMP-capable kernel by comparing the 1T-UP (one processor, uniprocessor kernel) value with the 1T-SMP (one processor, SMP kernel) value. The loss from uniprocessor kernel to SMP kernel on the base 3.06 GHz processor is 10% for Apache, and 13% for Flash and Tux. The losses on the L3-equipped processor and the 2.0 GHz processor are 14% for Apache and 18% for Flash and Tux, which are a little higher than our base system. The impact on Haboob is relatively low (4%-10%), because it performs the most non-kernel work. The magnitude of the overhead is fairly large, even though Linux has a reputation of being efficient for low-degree SMP configurations. This result suggests that, for uniprocessors, the performance gained from selecting the uniprocessor kernel instead of SMP kernel can be significant for these applications.

The fact that the impacts are larger for both the slowest processor and the processor with L3 are also interesting. However, if we consider these results in context, it can be explained. The extra overheads of SMP are not only the extra instructions, but also the extra uncacheable data reads and writes for the locks. The fastest system gets its performance boost from its L3 cache, which makes the main memory seem closer to the processor. However, the L3 provides no benefit for synchronization traffic, so the performance loss is more pronounced. For the slowest processor, the extra instructions are an issue when the processor is running at only two-thirds the speed of the others.

3.2 Uniprocessor SMT Benefits

Understanding the benefits of SMT for uniprocessors is a little more complicated, because it must be compared against a base case. If we compare 1T-SMP to 2T (uniprocessor SMT), the resulting graphs would appear to make a great case for SMT, with speedups in the 25%-35% range for Apache, Flash and TUX, as shown in Figure 4. However, if we compare the 2T performance

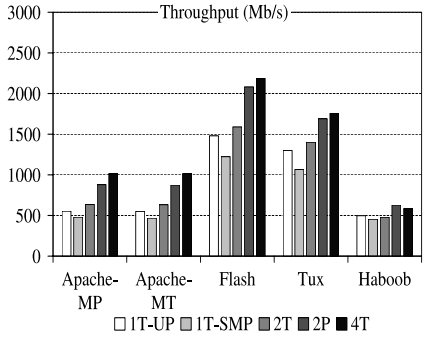

Figure 1: Throughput of Xeon 2.0GHz processor without L3 cache

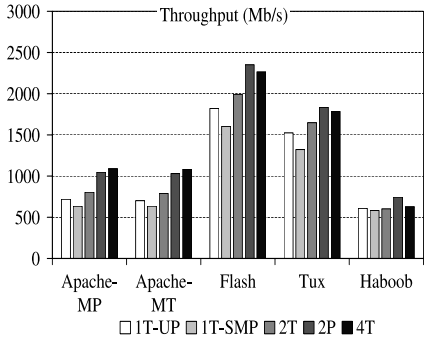

Figure 2: Throughput of base Xeon 3.06GHz processor

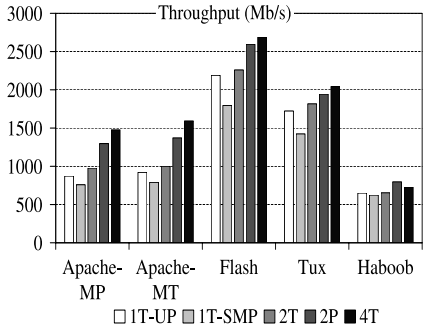

Figure 3: Throughput of Xeon 3.06GHz processor with 1MB L3 cache

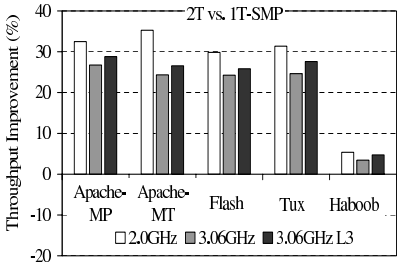

Figure 4: SMT speedup on uniprocessor system with SMP kernel

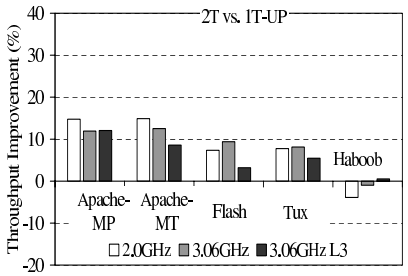

Figure 5: SMT speedup on uniprocessor system with different kernels

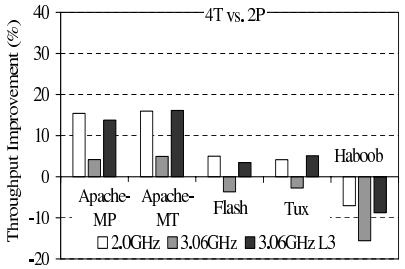

Figure 6: SMT speedup on dual-processor system

versus 1T-UP, then we see that the speedups are much more modest. These comparisons are shown in Figure 5, for all three processor types. In general, the relative gain decreases as processor becomes faster (via clock speed or cache). Apache-MT's gain on the 2.0 GHz processor is the highest at 15%, but this drops to the 10%-12% range for faster processors. The gains for Flash and TUX are less, dropping to the 3%-5% range for the faster processor. The Haboob numbers show the opposite trend from all other servers, showing a loss at 2.0 GHz improving to a small gain.

We believe that the correct comparison for evaluating uniprocessor SMT benefits is comparing the bandwidths with 1T-UP. Although the kernels are different, the SMP kernel needlessly hinders uniprocessor performance. For parallel algorithms, comparison with the best base case is also a standard speedup measurement technique. The performance of a parallel algorithm is compared to the performance of the best sequential algorithm. Simply put, the gain from choosing the appropriate kernel is comparable to the gain of upgrading hardware.

In comparing what is known about measured speedups from enabling the Xeon's SMT, our results are comparable to the 20%-24% gains that Tuck and Tullsen observed using other workloads [35]. Their speedup comparisons are performed using an SMP kernel for all measurements, which would be similar to comparing our 2T results to the 1T-SMP values. In fact, our observed speedups are slightly higher than theirs, if we discount Haboob. This result is in-line with the observation that SMT can potentially help server-style software more than other workloads [26]. The impact of using a uniprocessor kernel on the Tuck and Tullsen results is not clear – their workloads are not OS-intensive, so the performance loss of using an SMP kernel may be less than what we observed.

3.3 SMT in Dual-processor systems

The next reasonable point of comparison for SMT is in dual-processor systems, since these systems are particularly targeted to the server market and are rapidly approaching "commodity" status. Two factors responsible for this shift are the falling CPU prices and the support for low-degree multiprocessing built into some chips. The Xeon processors are available in two variants, the Xeon DP and the Xeon MP, with the distinguishing feature being the number of processors that can be used in one system. The DP ("dual-processor") line has on-chip support for building "glueless" 2-processor SMP systems that require no extra hardware to share the memory bus. The MP ("multiprocessor") line is intended for systems with more than 2 processors. In addition to the on-chip glue logic, the Xeon DP also drives commodification of dual-processor systems via pricing – as of this writing the Xeon DP is roughly one-tenth the cost of a Xeon MP at the same clock rate. Whether this difference stems from pricing strategies or economies of scale is unclear to us, but it does greatly magnify the price difference between dual-processor systems and larger multiprocessors.

We note that enabling SMT in a dual-processor configuration carries more risk than enabling it for uniprocessors. While the actual gains in uniprocessors may have been comparable to the loss in using an SMP kernel, the overall gains were still positive. As shown in Figure 6, enabling SMT in dual-processor configurations can cause a performance loss, even though the same kernel is being used in both cases. Haboob shows a 9%-15% loss on the three different processors, when comparing the 2P configuration to the 4T configuration. Flash and TUX show a loss in the base 3.06 GHz case, but show small gains for the other two processor types. The specifics of the performance curves lead us to believe that Flash

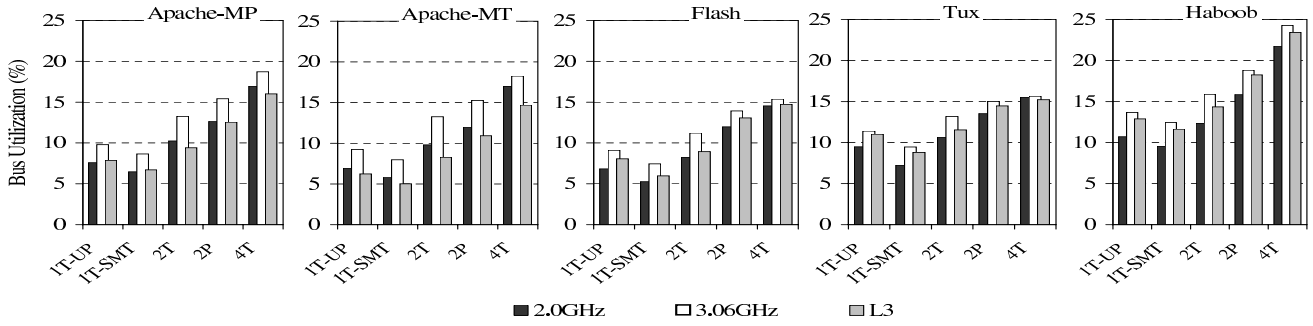

Figure 7: Bus utilization of three hardware configurations

and TUX are bottlenecked on the memory system – the base 3.06 GHz processor will have more memory traffic than its L3-equipped counterpart. Likewise, the 2 GHz processor has relatively faster memory, measured in CPU cycles, since the processor speed is slower. So, by taking a 2-processor system whose bottleneck is already memory, and increasing the memory demand, the overall performance will not improve. By the same reasoning, we can infer that Apache may be processor-bound, since it sees gains on all of the processors. The highest gain in this test, both in terms of absolute bandwidth and in percentage, is seen in the Apache-MT results for the L3-equipped 3.06 GHz processor. It gains 16% over the 2P configuration, jumping from 1371 Mb/s to 1593 Mb/s. The gain for Apache-MP on this processor is also significant, but smaller.

3.4 Understanding Relative Gains

These results are interesting because Apache is neither the best performer nor the worst – it appears to be in a "sweet spot" with respect to the benefits of SMT. This sweet spot may not be very large, in terms of the variety of configurations for which it works – Apache's gain on the base 3.06 GHz is only 4%-5%. Going forward, it may be necessary to keep increasing cache sizes to prevent faster processors from being bottlenecked on memory. If all of the contexts are waiting on memory, SMT may not be able to provide much benefit.

However, when using the relative gains, one should remember that they are compared only to the same server software, and may only reflect some artifact of that server. For instance, while Apache's relative gains are impressive, the absolute performance numbers may be more important for many people. Those show clearly that even the best Apache score for a given processor class never beats the worst Flash score, and almost never beats the worst TUX score. So, even with 2 SMT-enabled processors, Apache still does not perform as well as Flash (or TUX, generally) on a single processor.

3.5 Measuring the Memory Bottleneck

In the previous analysis, we have attempted to ascribe some of the performance characteristics of the various servers and configurations to their interaction with the memory system. To quantify these effects, we measure the cycles when the memory bus is occupied by any one of the threads, including both driving data onto or reading data from the bus. Even though the bus utilization figures do not differentiate "pointer chasing" styles of memory accessing from bulk data copying, by knowing the particular optimizations used by the servers, we can use this information to draw reasonable conclusions. To normalize the different processor clock speeds, bus utilization is calculated as follows:

$$Utilization = \frac{(CyclesBusOccupied) * (ClockSpeed)}{(NonHaltedCycles * BusSpeed)} \quad (1)$$

The bus utilization values, broken down by server software, configuration, and processor type, are shown in Figure 7. Several first-order trends are visible: bus utilization tends to increase as the number of contexts/processors is increased, is comparable for all servers except Haboob, and is only slightly lower for L3-equipped processors. The trends can be explained using the observations from the bandwidth study, and provide strong evidence for our analysis about what causes bottlenecks.

The increased bus utilization for a given processor type as the number of processors and hardware contexts increase is not surprising, and is similar in pattern to the throughput behavior. Essentially, if the system is work-conserving, we expect bus utilization to be correlated with the throughput level. In fact, we see this pattern for the gain from the 2.0 GHz processor to 3.06 GHz – the coefficient of correlation between the throughput and the bus utilization is 0.95. The coefficient for the L3-equipped versus base 3.06 GHz Xeon is only 0.62, which is still high, and provides evidence that the L3 cache is definitely affecting the memory traffic. A more complete explanation of the L3 results are provided below.

The fact that Haboob's bus utilization looks different from others is explained by its lack of zero-copy support, and in turn explains its relatively odd behavior in Figures 5 and 6. The bulk data copying that occurs during file transfers will increase the bus utilization for Haboob, since the processor is involved in copying buffers and performing TCP checksums. However, the absolute utilization values mask a much larger difference – while Haboob's bus utilization is roughly 50% higher than that of Flash or TUX, its throughput is one-half to one-third the value achieved by those servers. Combining those figures, we see that Haboob has a per-request bus utilization that is three to four times higher than the other servers.

The same explanation applies to the bus utilization for the L3-equipped processors, and to Apache's relative gain from SMT. The L3 cache absorbs memory traffic, reducing bus utilization, but for Flash and TUX, the L3 numbers are only slightly below the non-L3 numbers. However, the absolute throughput for the L3-equipped processors are as much as 50% higher, indicating that the per-request bus utilization has actually dropped. The differences in bus utilization then provide some insight into what is happening. For Flash and TUX, the L3 bus utilizations are very similar to the non-L3 values, suggesting that the request throughput increases until the memory system again becomes the bottleneck. For Apache, the L3 utilization is lower than the non-L3, suggesting that while the memory system is a bottleneck without the L3 cache, something

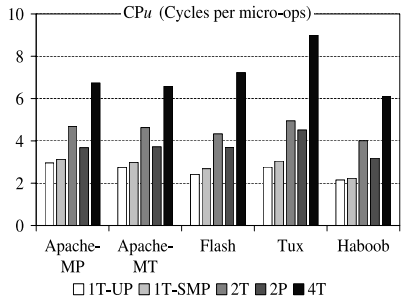

Figure 8: Cycles per micro-op (CPμ)

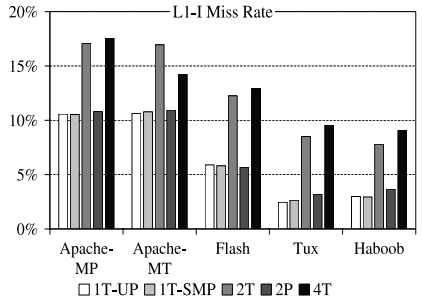

Figure 9: L1 instruction cache (Trace Cache) miss rate

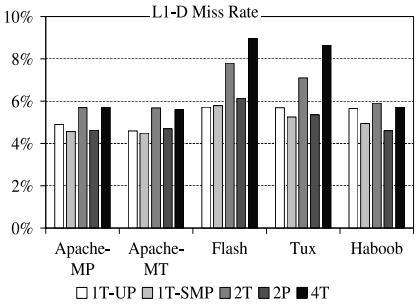

Figure 10: L1 data cache miss rate

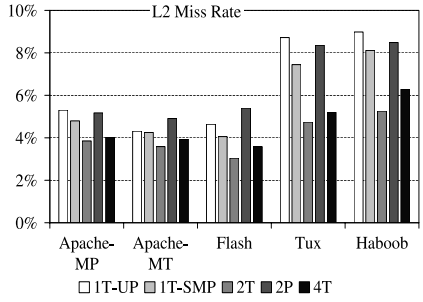

Figure 11: L2 cache miss rate, including both instruction and data

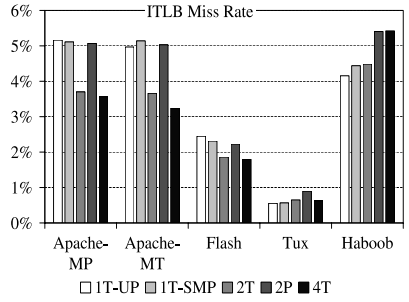

Figure 12: Instruction TLB miss rate

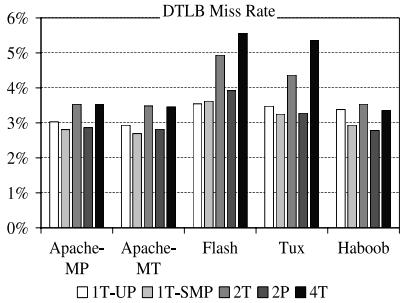

Figure 13: Data TLB miss rate

	Apache-MP	Apache-MT	Flash	Tux	Haboob
μPB	12.5	13.0	5.7	6.0	20.7
IPB	6.8	7.1	3.2	3.4	10.7

Table 4: Average Instructions and μops per byte for all servers

else becomes the bottleneck with it. Since we know the memory system is capable of higher utilization, we can conclude that CPU processing is the bottleneck. The explanation addresses both Apache's benefit with a second processor as well as with SMT enabled. Since Apache has more non-memory operations than Flash or TUX, it can benefit from the additional CPU capacity.

4. MICROARCHITECTURAL ANALYSIS

To understand the underlying causes of the performance we observed, we use the hardware events available on the Xeon. These events can monitor various microarchitectural activities, including cache misses, TLB misses, pipeline stalls, etc. While these counts discover low level resource utilization, their effects are hard to quantify since the Xeon's long pipeline overlaps many of these events' occurrence. We examine cycles per instruction and track various causes of these cycles spent. At a high level, CPU cycles can be modeled by three factors: cycles required to graduate the given instruction, instruction related stalls caused by unavailable instructions or data, and stalls due to pipeline resource limits. For instruction-related stalls caused by cache or TLB misses, we are able to calculate the individual numbers of cycles for each event type based on the known miss penalties. For pipeline resources, the Xeon only provides the number of stalls caused by allocator buffer shortages. Stalls from other sources, such as lack of other buffers or decode-execute interlocks, are not available.

We next describe a number of important metrics from a microarchitectural standpoint. We show measurements on our base 3.06 GHz processors since all of our processors have exactly the same resources such as cache size, TLB lines, buffers, etc., and we do not observe radical differences between them for the metrics we present here.

- **Cycles per instruction (CPI).** We measure the number of non-halted cycles and the number of instructions retired to calculate CPI. Since the Xeon decodes each instruction into multiple micro-ops (μops), we report CPI or CPμ where appropriate. The ratio of μops to instructions in our application ranges between 1.75 - 1.95. Figure 8 shows cycles per μop on each logical processor base for all of the servers. Network servers demonstrate much higher CPμ than other workloads, with the minimum value of 2.15, while the Xeon's optimal CPμ is 0.33, with three μops graduating per cycle.

  While CPI is commonly used for indicating processor execution quality or efficiency, it is not a perfect metric for some parts of our study. Because of the varying code bases, the number of instructions used to deliver a single byte of content also differs. For this reason, we may also report counts in terms of application-level bytes transferred, shown in Table 4 as instructions and μops per byte (IPB and μPB). We discuss some of the event results below.

- **Cache behavior.** In SMTs, the multiple contexts share all of the cache resources. This sharing may cause extra cache pressure because of conflicts, but may also reinforce each other. By comparing miss rates, we are able to detect whether cache conflicts or reinforcement dominates. Figures 9, 10, and 11 show miss rates for the L1 instruction (trace) cache, the L1 data cache, and the combined L2 cache, respectively.

  When SMT is enabled (in 2T and 4T), both the L1 instruction and data caches show significantly higher miss rates, indicating extra pressure, but the L2 miss rates improve, indicating benefits from

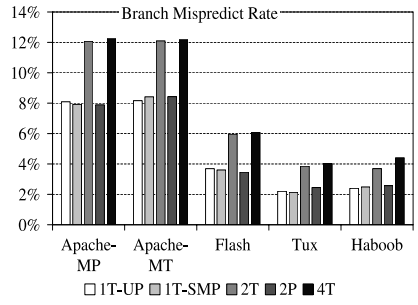

Figure 14: Branch misprediction rate

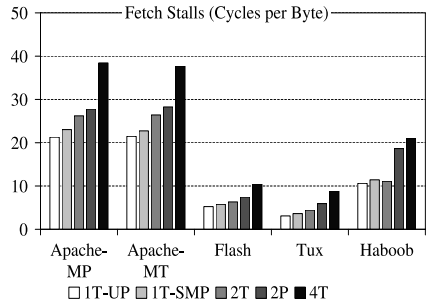

Figure 15: Trace delivery engine stalls

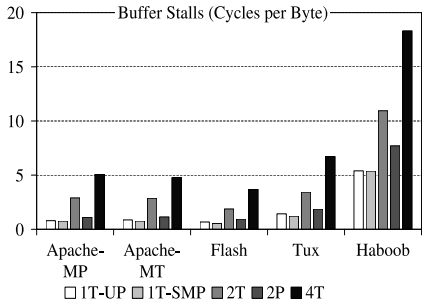

Figure 16: Stalls due to lack of store buffers

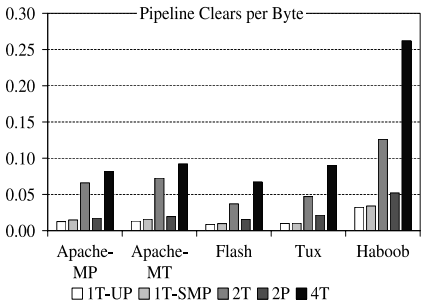

Figure 17: # of pipeline clears per byte

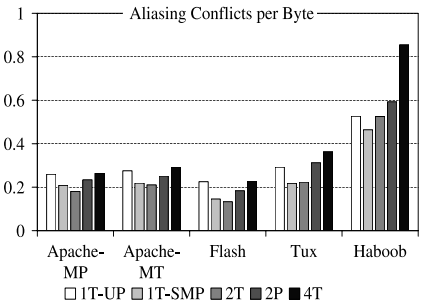

Figure 18: # of aliasing conflicts per byte

sharing. In comparing Apache-MT to Apache-MP, we do see some reduction in the 4T L1 miss rate, but the miss rate is still higher than the 2P cases. Thus, while the multithreaded code helps reduce the pressure, the SMT ICache pressure is still significant. The L2 miss rate drops in all cases when SMT is enabled, indicating that the two contexts are reinforcing each other. The relatively high L2 miss rate for TUX is due to its lower L1 ICache miss rate – in absolute terms, TUX has a lower number of L2 accesses. The interactions on CPI are complex – the improved L2 miss rates can reduce the impact of main memory, but the much worse L1 miss rates can inflate the impact of L2 access times. We show the breakdowns later when calculating overall CPI values.

- **TLB misses.** In the current Xeon processor, the Instruction Translation Lookaside Buffer (ITLB) is duplicated and the shared DTLB is tagged with each logical processor's ID. Enabling SMT drops the ITLB miss rate (shown in Figure 12) while increasing the DTLB miss rate (shown in Figure 13). The DTLB miss rate is expected, since the threads may be operating in different regions of the code. We believe the drop in ITLB stems from the interrupt handling code executing only on the first logical processor, effectively halving its ITLB footprint.

- **Mispredicted branches.** Branches comprise 15% - 17% of instructions in our applications. Each mispredicted branch has a 20 cycle penalty. Even though all of the five servers show 50% higher misprediction rates with SMT, the overall cost is not significant compared to cache misses, as we show in the breakdowns later.

- **Instruction delivery stalls.** The cache misses and mispredicted branches result in instruction delivery stalls. This event measures the number of cycles halted when there are no instructions ready to issue. Figure 15 shows the average cycles stalled for each byte delivered. For each server, we observe a steady increase from 1T-UP to 4T, suggesting that with more hardware contexts, the number of cycles spent stalled increases.

- **Resource Stalls.** While the value of instruction delivery stalls measures performance in the front-end of the pipeline, stalls may also occur during pipeline execution stages. This event measures the occurrence of stalls in the allocator caused by store buffer restrictions. In the Xeon, buffers between major pipeline stages are partitioned when SMT is enabled. Figure 16 shows cycles stalled per byte due to lack of the store buffer in the allocator. Enabling SMT exhibits a doubling of the number of stall cycles for each byte transfered. Unfortunately, stalls due to other buffer conflicts, such as the renaming buffer, are not available on existing performance-monitoring counters. We expect similar pressure is also seen in other buffers.

- **Pipeline clears.** Due to the Xeon's design, there are conditions in which all non-retiring stages of the pipeline need to be cancelled. This event measures the number of these flushes. When this happens, all of the execution resources are idle while the clear occurs. Figure 17 shows the average number of pipeline clears per byte of content. The SMT rate is a factor of 4 higher, suggesting that pipeline clears caused by one thread can affect other threads executing simultaneously. Profiling on this event indicates that more than 70% are caused by interrupts. Haboob's high clear rate in 4T mode may be responsible for some of its performance degradation.

- **64K aliasing conflicts.** This event occurs when the address of a load or store conflicts with another reference which is in progress. When this happens, the second reference cannot begin until the first one is evicted from the cache. This type of conflict exists in the first-level cache and may incur significant penalties for loads that alias to preceding stores. The number of conflicts per byte is shown in Figure 18. All of the servers show fairly high number of conflicts, suggesting an effective direction for further optimization.

- **Putting cycles together.** We estimate the aggregated cycles per instruction of these negative events and compare them to the measured CPI. While it is possible to estimate the penalty of each event,

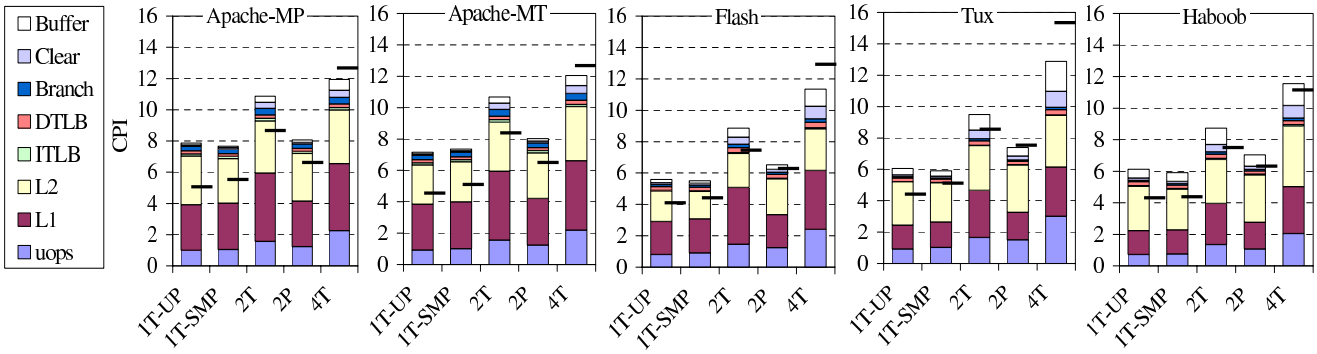

Figure 19: Non-overlapped CPI accumulated by cache miss, TLB miss, mispredicted branches and pipeline clear. Labels shown here such as L1, L2 etc. are misses, and components in each bar from top to bottom are in the same order as in the legend. Measured CPIs are shown as small dashes.

some have aggregated effects and thus are not included. Figure 19 shows breakdowns of non-overlapped CPIs calculated from eight events, with measured CPI shown as dashes. The breakdowns indicate that L1 and L2 misses are responsible for most of the cycles consumed. Pipeline clears and buffer stalls also have a significant portion when SMT is enabled, as shown in Flash, Tux and Haboob's 2T and 4T cases. Other events such as TLB misses and mispredicted branches are not major factors in our workloads.

Our microarchitectural analysis provides quantitative explanations of the observed performance and discovers a number of SMT resource bottlenecks. Quantifying performance change based on processor events for our-of-order superscalar processors is not our goal, nor do we think it is feasible. However, by examining the aggregated CPI and measured CPI, we can estimate the pipeline overlapping if the measured CPI is lower than the calculated value, and how many cycles are taken by other sources if measured CPI is higher. With this microscopic information and observed performance improvement, we can compare to similar studies using simulation which we describe in the next section.

5. EVALUATING THE SIMULATIONS

Our measurements present a much less optimistic assessment of SMT performance benefits than many of the simulation-based studies – most of our gains are in the 5%-15% range for 2 threads, while the simulations show speedups in the 200%-400% range for 4-8 threads. Intuitively, the number of threads might be the cause of this speedup gap. However, studies also show that the first few threads usually exhibit more performance gain than the rest [36]. Thus, the simulated speedup for 2 threads would be in the range of 70-100%, which is still much higher than what we observe. While none of the published simulations modeled the Xeon, the significant disparity in the gains warrants analyzing their cause. We have not found any simulations specifically regarding multiprocessor systems, although such systems are popular in the network server market. None of the simulations appear to have considered the cost of using an SMP-enabled kernel instead of a uniprocessor kernel, and we have shown this cost to be significant. However, we believe the other significant differences are hardware-related, which we discuss below.

The most prominent area of difference between the Xeon and the simulated processors is the structure of the memory hierarchy, and the associated latencies. The Xeon has an 8KB L1 data cache and a 12K μops trace cache (TC), which is equivalent to an 8KB - 12KB conventional instruction cache. Detailed hardware parameters and latencies for our experimental platform are presented in Table 5. The 1996 study is a proposal for practical SMT processors [36], while the 2000 paper examines SMT OS and Web server performance [26], and the 2003 one examines SMT search engine performance [18]. The processor models were derived from Alpha, and have shorter pipelines and slower clock speeds than modern processors. While the 1996 design had caches that are comparable to what is available today, the others are much more aggressive than what is currently available.

Type	sim	sim	sim	Xeon	Xeon
Year	1996	2000	2003	2003	2004
Clock rate (Mhz)	600*	800*	800*	2000	3060
# Contexts	8	8	8	2	2
# Stages	9	9	9	20	30
L1 ICache (KB)	32	128	64	8*	8*
L1 DCache (KB)	32	128	64	8	8
L2 size (KB)	256	16384	16384	512	512
L2 cycles	6	20	20	18	18
L3 size (KB)	2048	-	-	-	1024
L3 cycles	12	-	-	-	46
Memory (cycles)	62	90	90	225	344

Table 5: Processor parameters in simulation and current products. Values marked with an asterisk are approximate or derived.

The issue of cache size is significant, because of its direct impact on processor cycle time. Larger caches slow access times, and the Xeon's L1 caches are small in order to support its high clock frequencies. For comparison, if we triple the clock frequencies, stages, and main memory latencies for the 2000 and 2003 studies, then those values are in line with the current Xeons, but the L1 cache sizes are 8-16 times higher, and the L2 cache size is 32 times larger. If we assume the simulated L2 caches are really L3, then they are more than twice as fast as the Xeon's L3 latencies, while still being 4 times larger than the L3 caches of any Xeon in the market at the time this study was performed. Even if we compare with high-end processors, the simulated processors are still aggressive. For example, the IBM POWER5 has a 64KB instruction and 32 KB data L1 cache, 1.9MB L2 cache, 36MB of shared L3 cache per 2 processors, 2 SMT contexts, a 16-stage pipeline, and operates at 1.5 GHz [11]. Compare to the scaled version of the simulated processors, it has one-fourth the SMT contexts, and operates at half of the clock speed. We conclude that not only are the simulated mem-

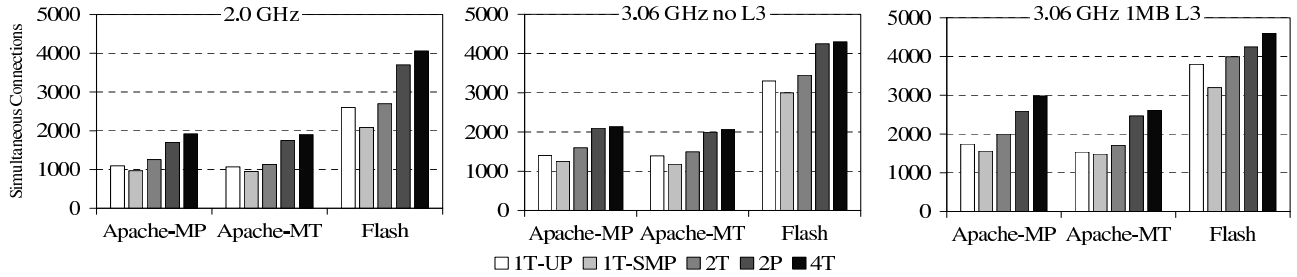

Figure 20: SPECweb99 scores of three servers. The metric is number of simultaneous connections.

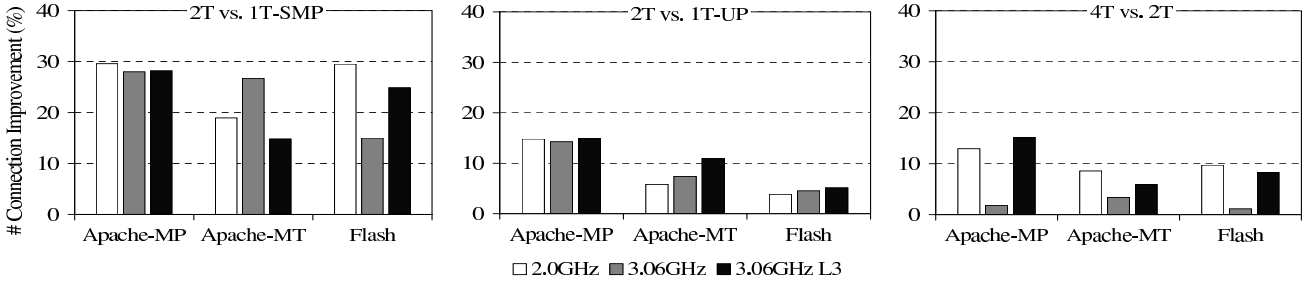

Figure 21: SMT speedups on SPECweb99 scores for the three servers.

ory hierarchies more aggressive than what current hardware can support, but the memory latencies are also much faster than what might be reasonably expected from their size. Given the sensitivity to cache sizes and memory speeds we have seen in our evaluations, it is not surprising that the simulations yield more optimistic speedup predictions.

While these differences indicate that SMT in the Xeon may not yield significant benefit on the workloads we studied, it does not imply that SMT in general is not useful. If a company were to design a processor specifically suitable for SMT, it may choose a larger number of contexts and larger caches while consciously sacrificing cycle time. Such a system may have poor performance for single-threaded applications, but would be suitable for highly-parallel tasks. Sun Microsystems has discussed their upcoming "Niagara" processor, which has 8 cores with 4 contexts per core [23]. They project it to have 15 times the performance of today's current processors, while a dual-core UltraSparc V microprocessor targeted for the same timeframe was expected to have 5 times the performance of today's processors. Using these numbers, each context on Niagara will perform at one-fifth the performance of one UltraSparc V context. This approach is similar to the Denelcor HEP [24] and the Tera MTA [30] which were designed for high throughput instead of high single-thread performance. Whether this approach will be more successful for a higher-volume processor remains an open question.

In broader terms, though, the simulations identified a number of trends that we can confirm via our evaluation. Table 6 compares our measured results versus the simulation study of the same workload [26]. While the magnitude of the values between the simulated and actual processor is large due to the differences in cache sizes, etc., the direction of change is the same for each metric.

	Simulation		Measurement	
# Contexts	8		2	
Speedup	4-fold		5-15%	
	SMT	ST	SMT	ST
IPC	5.6	2.6	0.43	0.33
Branch Mispredict (%)	9.3	5.0	12.0	8.0
L1-I miss (%)	2.0	1.3	17.1	10.5
L1-D miss (%)	3.6	0.5	5.7	4.7
L2 miss (%)	1.4	1.8	3.9	5.1
ITLB miss (%)	0.0	0.0	3.7	5.1
DTLB miss (%)	0.6	0.05	3.5	2.9

Table 6: Results comparison between related simulation work and our measurements. We use ST (Single Threaded) to indicate the non-SMT performance.

6. SMT ON SPECWEB99 BENCHMARK

While static Web workloads are useful to compare with previous studies, dynamic content is an important part of current Web traffic, and is captured in the SPECweb99 benchmark. In order to compare with results discussed in previous sections, we run the full benchmark but also limit the data set size to 500 MB.

SPECweb99 introduces changes to both the workload and methodology of the SPECweb96 benchmark. The benchmark consists of 70% static and 30% dynamic requests. The dynamic requests attempt to model commercial Web servers performing ad rotation, customization, etc., and require some computation. Rather than reporting rates in requests per second, SPECweb99 reports the number of simultaneous connections the server can handle while meeting a specified latency requirement.

The dynamic portion of SPECweb99 consists of a specification which must be implemented for each server. Because we do not have versions of this specification for all of our servers, we can only evaluate it for three of our five systems. We run Flash as well as both versions of Apache on the three hardware configurations. While TUX has SPECweb99 scores, we have been unable to get the dynamic content support to work on the free version of Linux with SMT support. We are investigating its performance on the Red Hat Enterprise distribution since it is more stable and is the distribution for which most TUX results are reported. Haboob is not included because we did not find its dynamic API for SPECweb99. Since we focus on the differences stemming from SMT, we are not

overly concerned about the missing servers. For Apache, we use the mod_specweb99 module which is available on the Apache Web site. Similarly, for Flash, we use its built-in SPECweb99 module that handles dynamic requests.

Figure 20 shows the SPECweb99 scores of the three servers on the three different hardware configurations, and Figure 21 calculates the speedups. The trends are generally consistent with what we observed in Section 3, but some interesting differences emerge. On uniprocessor systems, SMT has a speedup of 4% to 15% when comparing to the uniprocessor kernel, and a speedup of 15% to 30% when comparing to the SMP kernel. On dual-processor systems, the improvements range from 1% to 15%. In comparing Figure 21 with Figures 5 and 6, we see some interesting differences. The gains for Apache on SpecWeb99 are almost always worse than those on the static-only test. Flash's gains are worse at one processor, but better at two processors. The additional computation in SpecWeb99 appears to help utilize the processor better when the memory system is very taxed, but seems to be causing more imbalance in Apache, which already does more computation than Flash.

This result also leads to another general observation. While SMT is intended to hide memory reference latency by overlapping the use of idle processor resources, applications threads having similar resource utilization characteristics such as Web servers serving static content may have little to overlap. Using dissimilar workloads to achieve better SMT utilization has been explored for the CINT (integer component of SPEC CPU) benchmark programs [33], but not for network server software. Even with this mix of programs, the benefits we see in the network server environment are lower than in the CPU-only tests.

7. RELATED WORK

In this section we discuss related work not already covered. When investigating SMT performance, operating systems were not included in simulations until Redstone et al. [26] ported the SMTSIM simulator [37] into the SimOS framework [27]. They discovered that although ignoring operating systems behavior may not result in misleading predictions for SPEC CINT, it has significant impact on evaluation of server applications such as Apache. McDowell et al. [18] used the same simulator and studied memory allocation and synchronization strategies for a search engine application. Similarly, many studies focus on user-level and compute-intensive applications, including SPEC CINT and CFP [32, 36, 37], parallel ray-tracing applications [10], SPLASH-2 benchmarks [16], MPEG-2 decompression [7, 29], and other scientific application workloads [13, 31]. Lo et al. [15] analyzed SMT performance with database workloads, which spend 70% of their time in user space.

Our approach differs from previous performance evaluations in several ways. While direct measurement on real hardware gives accurate results and does not need model validation, as is required by simulation, it is limited by the available hardware configurations. By making small changes to the hardware and exploring kernel options, we obtain a reasonable space for comparison. More importantly, compared to the work which studies server applications, our evaluation has a broader range of server software, OS support and hardware configurations. Our results on uniprocessor kernels and dual-processor systems discover new SMT performance characteristics. In contrast to other works, we focus on server workloads since it is one of the biggest markets for SMT-enabled processors.

Performance evaluation and characterization on currently available SMT-enabled processors is still an ongoing research area. Tuck and Tullsen [35] evaluated the implementation and effectiveness of SMT in a Pentium 4 processor, particularly in the context of prior published research in SMT. They measured SPEC CPU2000 and other parallel benchmarks, and concluded that this processor implementation matches the promise of the published SMT research. Bulpin and Pratt [6] extended Tuck et al.'s evaluation by comparing an SMT system to a comparable SMP system, but did not investigate SMT on SMP systems as we do in this paper. Chen et al. [7] also only evaluated performance of SMT and SMP individually. Vianney [39] measured a single Xeon processor and reported that Hyper-Threading on the Linux kernel can improve throughput of a multi-threaded application (namely, chat [14]) as much as 60%. Our study not only differs in target testbed and workloads, but also provides low level microarchitectural characteristics to reveal detailed resource utilization pertinent to SMT.

SMT's microarchitectural performance is always one of the main concerns in SMT design. In addition to work discussed earlier in this section, Grunwald and Ghiasi [9] examined the Xeon processor and discovered a possible microarchitectural denial of service attack for the SMT processor. Snavely et al. [31, 32] observed symbiotic features in SMT architectures and proposed a special scheduler to exploit it. Raasch and Reinhardt [25] studied the impact of resource partitioning on SMT processors. Our microarchitectural analysis using the performance counters focuses on the comparison between when SMT is enabled and disabled, instead of evaluating the performance of different SMT design options.

Performance analysis using hardware provided event counters has been an effective approach in previous studies. Bhandarkar et al. [4] and Keeton et al. [12] characterized performance of Pentium Pro systems and studied latency components of the CPI. More recently, Blackburn et al. [5] used some of the Pentium 4 performance counters to study impact of garbage collection. Given the complexity of Xeon microarchitecture, interpreting the performance-monitoring events on these systems is more difficult than with previous Pentium family processors or RISC processors. Moreover, we are unaware of any non-simulation work in this area that provides the breadth of event coverage that we do in this paper.

8. CONCLUSION

This paper provides a performance analysis of simultaneous multithreading for server applications, using five software implementations and three hardware platforms. We find that the performance benefits of SMT are much more modest when compared to the uniprocessor kernel, suggesting non-negligible amounts of OS overhead when supporting SMT. This cost was mostly ignored in previous studies. Our evaluation in dual-processor configurations indicates that the benefits of SMT are harder to achieve in these systems, unless memory reference latency is shorter or a large L3 cache is used, a finding that may aid SMT design and purchasing.

Our microarchitectural analysis provides quantitative explanations of the observed performance and discovers a number of SMT resource bottlenecks. Using this information, future processor designers can understand how to better serve this important class of applications. With this detailed, low-level information and observed performance improvement, we are able to compare our results to similar studies performed using simulation. We believe that simulation correctly predicts the direction of change for processor resources, but yields much more optimistic estimates of resource contention and overall speedup.

Acknowledgments

This work was partially supported by an NSF CAREER Award. We would like to thank our shepherd, Geoff Voelker, and our anonymous reviewers for their feedback and insight.

References

[1] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S. Leung, D. Sites, M. Vandevoorde, C. Waldspurger, and W. Weihl. Continuous profiling: Where have all the cycles gone. In *Proc. of the 16th SOSP*.

[2] Apache Software Foundation. The Apache Web server. http://www. apache.org/.

[3] P. Benmowski. Hyper-Threading Linux. *LinuxWorld*, Aug. 2003.

[4] D. Bhandarkar and J. Ding. Performance characterization of the Pentium Pro processor. In *Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA '97)*, Feb. 1997.

[5] S. Blackburn, P. Cheng, and K. McKinley. Myths and realities: The performance impact of garbage collection. In *Proc. of the SIGMETRICS '04*, June 2004.

[6] J. Bulpin and I. Pratt. Multiprogramming performance of the Pentium 4 with Hyper-Threading. In *Workshop on Duplicating, Deconstructing, and Debunking (WDDD04)*, June 2004.

[7] Y.-K. Chen, E. Debes, R. Lienhart, M. Holliman, and M. Yeung. Evaluating and improving performance of multimedia applications on simultaneous multi-threading. In *9th International Conference on Parallel and Distributed Systems*, Dec. 2002.

[8] S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm, and D. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, Sept. 1997.

[9] D. Grunwald and S. Ghiasi. Microarchitectural denial of service: insuring microarchitectural fairness. In *Proc. of the 35th International Symp. on Microarchitecture*, 2002.

[10] H. Hirata, K. Kimura, S. Nagamine, Y. Mochizuki, A. Nishimura, Y. Nakase, and T. Nishizawa. An elementary processor architecture with simultaneous instruction issuing from multiple threads. In *Proc. of the 19th ISCA*, 1992.

[11] R. Kalla, B. Sinharoy, and J. M. Tendler. IBM Power5 chip: A dual-core multithreaded processor. *IEEE Micro*, March 2004.

[12] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker. Performance charaterization of a Quad Pentium Pro SMP using OLTP workloads. In *Proc. of the 25th ISCA*.

[13] H. Kwak, B. Lee, A. Hurson, S.-H. Yoon, and W.-J. Hahn. Effects of multithreading on cache performance. *IEEE Trans. Comput.*, 48(2), 1999.

[14] Linux Benchmark Suite Homepage. A GPL'd chat room benchmark. http://lbs.sourceforge.net/.

[15] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, H. Levy, and S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proc. of the 25th ISCA*.

[16] J. Lo, J. Emer, H. Levy, R. Stamm, and D. Tullsen. Converting thread-level parallelism to instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, 15(3).

[17] D. Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1).

[18] L. McDowell, S. Eggers, and S. Gribble. Improving server software support for simultaneous multithreaded processors.

[19] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *USENIX 1996 Annual Technical Conference*.

[20] OProfile. A system profiler for Linux. http://oprofile.sourceforge.net/.

[21] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *USENIX 1999 Annual Technical Conference*.

[22] V. Pai, P. Druschel, and W. Zwaenepoel. IO-Lite: a unified I/O buffering and caching system. *ACM Transactions on Computer Systems*, 18(1).

[23] G. Papadopoulos and D. Yen. Throughput computing: Driving down the cost of network computing. http://www.sun.com /events/analyst2003/presentations/Papadopoulos_Yen_WWAC_022503.pdf.

[24] R. E. Hiromoto, O. M. Lubeck, and J. Moore. Experiences with the Denelcor HEP. In *Parallel Computing*.

[25] S. Raasch and S. Reinhardt. The impact of resource partitioning on SMT processors. In *12th International Conference on Parallel Architectures and Compilation Techniques (PACT'03)*, New Orleans, Louisiana, Sept. 2003.

[26] J. Redstone, S. Eggers, and H. Levy. An analysis of operating system behavior on a simultaneous multithreaded architecture. In *Proc. of the 9th ASPLOS*.

[27] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta. Complete computer system simulation: The SimOS approach. *IEEE parallel and distributed technology: systems and applications*, 3(4), Winter.

[28] Y. Ruan and V. Pai. Making the "Box" transparent: System call performance as a first-class result. In *USENIX 2004 Annual Technical Conference*, Boston, MA, June 2004.

[29] U. Sigmund and T. Ungerer. Memory hierarchy studies of multimedia-enhanced simultaneous multithreaded processors for mpec-2 video decompression. In *Workshop on Multi-Threaded Execution, Architecture and Compilation*, January 2000.

[30] A. Snavely, L. Carter, J. Boisseau, A. Majumdar, K. S. Gatlin, N. Mitchell, J. Feo, and B. Koblenz. Multi-processor performance on the tera mta. In *Proc. of the 1998 ACM/IEEE conference on Supercomputing*, 1998.

[31] A. Snavely and D. Tullsen. Symbiotic job scheduling for a simultaneous multithreaded processor. In *Proc. of the 9th ASPLOS*, 2000.

[32] A. Snavely, D. Tullsen, and G. Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *Proc. of the SIGMETRICS'02*.

[33] Standard Performance Evaluation Corporation. http://www.spec.org/benchmarks.html.

[34] Standard Performance Evaluation Corporation. SPEC Web Benchmarks. http://www.spec.org/web99/ http://www.spec.org/web96.

[35] N. Tuck and D. Tullsen. Initial observations of a simultaneous multithreading processor. In *12th International Conference on Parallel Architectures and Compilation Techniques (PACT'03)*, Sept. 2003.

[36] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proc. of the 23th ISCA*.

[37] D. Tullsen, S. Eggers, and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc. of the 22th ISCA*, 1995.

[38] TUX Web Server. http://www.tux.org/.

[39] D. Vianney. Hyper-Threading speeds Linux. *IBM developerWorks*, Jan. 2003.

[40] M. Welsh, D. Culler, and E. Brewer. SEDA: An architecture for well-conditioned, scalable internet services. In *Proc. of the 18th SOSP*, Oct.