On the Effectiveness of Simultaneous Multithreading on Network Server Workloads

Yaoping Ruan[1] Erich Nahum[1] Vivek Pai[2] John Tracey[1]

IBM T.J. Watson Research Center [1]
Hawthorne, NY, 10532
{yaoping.ruan,nahum,traceyj}@us.ibm.com

Princeton University [2]
Princeton, NJ, 08540
vivek@cs.princeton.edu

Abstract

This paper experimentally investigates the effectiveness of simultaneous multithreading (SMT) for network server workloads. We study how well SMT improves performance on two very different shipping platforms that support SMT: IBM's POWER5 and the Intel Xeon. We use the architectural performance counters available on each processor to compare their architectural behavior, examining events such as cache misses, pipeline stalls, etc. By observing how these events change in response to the introduction of threading, we can determine whether and how SMT stresses the system.

We find that POWER5 makes more effective use of SMT for improving performance than the Xeon. In general, POWER5 achieves a 40-50% increase whereas the Xeon exhibits only a 10-30% gain. Examination using the performance counters reveals that cache size and memory bandwidth are the key requirements for fully exploiting SMT. A secondary but still noticeable component is minimizing pipeline stalls due to branch mispredicts and interrupts. We present suggestions for improving SMT performance for each platform based on our observations.

1 Introduction

Simultaneous multithreading (SMT) is a technique for improving processor resource utilization by allowing two or more hardware threads to execute simultaneously on a single processor core. By duplicating a small set of hardware resources, such as architectural state registers, SMT allows multiple instruction streams to execute in a single pipeline simultaneously. Because the additional hardware support for these extra resources is small, SMT is seen as a relatively inexpensive way to gain instruction throughput. Academic research has demonstrated considerable performance gains using this technique [11, 44]. However, since the actual hardware did not exist at the time of these studies, they could only be conducted using simulation.

This limitation has been changing, however, as SMT-enabled hardware has come to market. Industry acceptance of SMT has been gaining, as shown by recent shipping processors from major manufacturers such as IBM and Intel, who have promoted SMT-enabled processors for use in high performance servers. The availability of this hardware offers the opportunity to measure performance and see how effective SMT is in practice. However, current implementations of SMT have evolved from existing single-threaded processors, and thus competing manufacturers may have made very different choices regarding what resources need to be replicated or modified for supporting SMT. For example, the Intel Xeon processor with Hyper-Threading is based on the Pentium 4 NetBurst architecture, and the IBM POWER5 is derived from POWER4. Thus, studying the performance and observable bottlenecks in different processor families presents us with an excellent opportunity to understand what architectural features are important for supporting SMT.

This paper experimentally investigates the effectiveness of simultaneous multithreading for network server workloads. We study how well SMT improves performance on the two shipping platforms that support SMT: IBM's POWER5 and the Intel Xeon. We focus on server workloads since the processors are targeted for such usage and because most previous studies have examined user-space and/or CPU-bound workloads. We measure the performance of two different Web servers, Apache [2] and Flash [30], running on a standard client-server testbed connected using an isolated switched Gigabit Ethernet, using the industry-standard SPECWeb99 [41] for workload generation.

In particular, we measure how performance changes with the introduction of SMT, thus evaluating how effective each processor at utilizing this feature. To help understand when, how, and why SMT is effective, we use the architectural performance counters available on each platform to measure and compare architectural behavior. By observing how cycle-intensive architectural events such as cache misses, ITLB misses, pipeline stalls, and branch

mispredictions change in response to adding SMT, we can determine whether and how SMT stresses the system. This paper makes the following contributions:

- We provide the first experimental performance comparison of SMT implementations across multiple platforms using server workloads. While POWER5 achieves 37-60% performance increases with SMT, the Xeon gains only 5-30%. The Xeon also exhibits less benefit from SMT in the dual-processor case than with a single processor. POWER5 improvements are more consistent for both single and dual processors.
- We show that cache provisioning is a crucial factor for exploiting SMT. Performance counter measurements indicate that the Xeon does not significantly benefit from SMT because of increased CPI due to cache contention. In contrast, cache and memory system contribution to overall CPI is only slightly affected by SMT on POWER5, which has much larger caches.
- Pipeline management is a secondary but significant factor for achieving SMT benefits. CPI is only slightly increased by pipeline stalls and branch mispredicts on the Xeon when SMT is enabled, but rises more significantly on POWER5.
- We provide recommendations for both platforms on how each can better take advantage of SMT.

The remainder of the paper is organized as follows. Section 2 provides some architectural background on hardware multithreading, particularly SMT. Section 3 describes our experimental platform in detail, including server hardware, network connectivity, server software, and workload generation. Section 4 presents the overall performance results of the two platforms. Section 5 delves further by using performance counter measurements to help understand the performance seen in Section 4. Section 6 discusses related work, and Section 7 presents our conclusions and directions for future work.

2 Background

In this section, we present a brief overview of the various approaches to multithreading, describe SMT in more detail, and discuss the two SMT-capable processors that we use – IBM POWER5 and Intel Xeon processor with Hyper-Threading.

2.1 Hardware Multithreading

Although SMT is a relatively new feature, the concept of multithreading has been around for several years. In these architectures, instructions from different threads are executed at various intervals without context switches. The execution interval could be interleaved cycle by cycle, known as *fine-grained multithreading*, or by certain events (such as a cache miss), known as *coarse-grained multithreading*. An early example of the former was the Denelcor HEP [32], which had 8 hardware contexts. A much later design from the same architect, the Tera MTA [10], used 128 contexts. Recent examples include the IBM RS64-II [6], introduced in 1998, which switches threads to avoid processor time during long-latency events. The Sun Fire T1000 (Niagara) from Sun Microsystems has fine-grained multithreading which implements a thread-select logic to decide which thread is active in a given cycle [31].

	Xeon	POWER5
Shared Resources	caches, branch predictors, decoder logic, execution units	
	DTLB	TLB
Duplicated Resources	interrupt controller, status registers, renaming logic	
	ITLB	instruction buffer,
Partitioned Resources	load/store buffer	
	μop queue, reordering buffer	Global completion table (GCT)

Table 1: SMT resource allocation in Intel Xeon and POWER5.

2.2 Simultaneous Multithreading

Multithreading technology in general was introduced to increase functional unit utilization. SMT improves this utilization by executing instructions from multiple processes or threads simultaneously in a single pipeline. In SMT-enabled processors, most of the processor resources are shared by the threads. Duplication of components varies from processor to processor, except that registers for architectural states have to be duplicated to fetch instructions from different processes independently. Because operating systems usually get processor information from these registers, SMT appears exactly as multiprocessor if no further identification is taken.

2.3 Intel Xeon

The Intel Xeon with Hyper-Threading was the earliest SMT-capable processor family shipped with real implementation. The Xeon is Intel's server class processor, with dual or multi-processor support. Intel Xeon with Hyper-Threading was based on the Intel NetBurst microarchitecture [14], which has a superscalar, out-of-order pipeline with 20 stages for early versions and 31 stages for the latest ones. It has two threads (logical processors) per physical processor, which share most of the resources, such as caches, functional units, branch prediction unit etc. Some of the resources are duplicated, such as architecture state registers, interrupt controllers, etc. Some resources

	Xeon	POWER5
Clock Rate	3.06 GHz	1.5 GHz
Pipeline Stages	30	16-21
TC/I-L1	12 K μops, 8-way, 6μops line	64 KB, 2-way, 128B line
D-L1	8 KB, 4-way, 64B line	32 KB, 4-way, 128B line
L2	512 KB, 8-way, 128B line	1.875 MB*, 10-way, 128B line
L3	1 MB, 8-way, 128B line	36 MB*, 12-way, 256B line
Main Memory	2 GB	2 GB
Memory Bandwidth	600 MB/s	1280 MB/s
Address Translation	128-entry ITLB 64-entry DTLB	1024-entry I/DTLB 128-entry ERAT 64-entry I/DSLB
Branch Prediction	4K branch target buffer	8-entry branch information queue

Table 2: Intel Xeon and POWER5 hardware specification. Memory bandwidth are measured using lmbench. *: the L2 and L3 caches are shared by the two cores on the POWER5

are partitioned when SMT is enabled, but are completely available to a single thread when only one thread is active [22]. We provide some of the key resource allocations for the Xeon in Table 1.

2.4 IBM POWER5

The POWER5 processor is IBM's latest 64-bit implementation of the PowerPC AS architecture [17]. It is a dual core processor with multi-threading technology. The processor has a superscalar, out-of-order pipeline with 16 stages for fixed-point operations, 18 stages for most load and store operations, and 21 stages for most floating point operations. Each POWER5 core supports two hardware threads. As with the Xeon, threads in POWER5 also share functional units and branch prediction logic. Replicated resources include state registers, the instruction buffer, etc. The design of the instruction buffer is very different in the two processors – it is replicated in POWER5 while is partitioned in Xeon (the μops queue). The reordering unit, which is achieved using a global completion table (GCT) in POWER5, is partitioned in both processors. Some of the hardware resources are also described in Table 1.

When comparing these platforms, we focus on the architectural components enhanced to support SMT. Direct architectural comparison between Xeon and POWER5 may not be feasible since Xeon is a complex instruction set computer (CISC) and POWER5 is a reduced instruction set computer (RISC). Instructions on these two processors are thus very different. However, the native x86 instruction is internally translated into RISC-like micro-operations (μops) which are comparable to instructions in POWER5.

Term Used	Kernel Type	Num Proc.	Threads / Proc.	Total Threads
UP	UP	1	1	1
1P	SMP	1	1	1
2T	SMP	1	2	2
2P	SMP	2	1	2
4T	SMP	2	2	4

Table 3: Nomenclature used in this paper

3 Experimental Testbed

Our experimental setup consists of an Intel Xeon server and an IBM POWER5 server. Our POWER5 server is a P5 520, which has a dual-core processor running at 1.5GHz. Our Intel server is a SE7505VB2 which supports Xeon dual processors running at a number of clock speeds. To simplify the comparison with the POWER5, we use a dual Xeon processor running at 3.06GHz with a 1MB L3 cache. Some of the detailed hardware parameters are provided in Table 2.

There are a few key differences in the memory hierarchy between the two processors. In POWER5, the L3 cache is off-chip, but moved from the memory side of the fabric to the processor side in order to reduce interchip traffic when an L2 miss hits in the L3. The Xeon only provides L3 cache for its high-end processors, but it is on-chip. The two processor cores in POWER5 share all of the caches. When only one processor is active, it is able to access the full cache capacity.

In addition to the two server systems, our testbed has 12 client workload generator machines with 1.6 GHz AMD Duron processors. We ensure that the aggregate processor power of the clients are enough to saturate the servers. To provide adequate network bandwidth, the clients are partitioned into four groups of three machines. Each group is connected to the server via a separate Gigabit Ethernet switch, which in turn is connected to one of four Intel e1000 MT server adapters on the server.

In our experiments, we compare five different OS/processor configurations, based on whether SMT is enabled or not and how many physical processors are active. The five configurations are one processor without SMT with a uniprocessor kernel (UP), one processor without SMT and an SMP kernel (1P), one processor with SMT (2T), two processors without SMT (2P), and two processors with SMT (4T). Our terminology is summarized in Table 3.

We use the BIOS support and OS boot parameters to enable and disable SMT for the Xeon processor. POWER5 provides a hot-plug interface to the OS in order to bring up and shut down a particular thread without requiring a

reboot. Since our goal is to evaluate SMT performance, we use the multiprocessor-capable (SMP) kernel, which is required to use SMT. While some previous work [36] finds that SMP kernels may reduce some of the benefits of SMT when comparing performance with uniprocessor kernel, that is not the focus this work. We use RedHat Enterprise Linux AS 4 (RHEL AS 4) on both of the servers. The kernel of RHEL 4 is a patched version 2.6.9 which includes optimizations for SMT such as processor affinity and load balancing [3]. The scheduler eliminates the chance of one context being idle while the other has multiple tasks waiting.

3.1 Workloads

We focus on Web servers and workloads since it is one of the most popular commercial server applications and has not been subject to the same level of study as scientific computing workloads. We use Apache 2.0 [2] and the Flash [30] Web servers. To exploit thread-level-parallelism, we configure the Apache server with the thread pool model which uses Linux kernel threads. Flash is an event-driven server with aggressive performance optimizations. It has a main process multiplexing all of the requests while all disk I/O accesses are handled by a set of helper processes. We run the same number of Flash main processes as the number of hardware contexts. All of the servers are optimized for the best performance by following steps described in literature. We disable logging on both of the servers to reduce the amount of disk activity.

We use the SPECweb99 [41] benchmark which measures server capacity in the number of simultaneous connections that meet a QoS target. The benchmark consists of 70% static and 30% dynamic requests. The dynamic requests attempt to model commercial Web servers performing ad rotation, customization, etc. The benchmark client software reports connections meeting the specified latency requirement. The data set involved scales with the number of connections, thus it is easily exceed the physical memory size. In order to avoid any complexity introduced by disk access, we limit the data set size to 500 MB, which fits in our physical memory.

When measuring performance counters, we use Oprofile [29] on Xeon and pmcount on POWER5. Oprofile support for POWER5 has been recently ported; however, we observe a significant amount of overhead when using Oprofile on POWER5. The overhead on Intel platform is minimal, usually less than 1%.

4 SMT Performance Results

In this section we evaluate the effectiveness of SMT by examining the performance of our two platforms. Here performance is not only the achieved throughput or number of simultaneous connections in SPECWeb99 or any other workloads, but also the *relative* improvement of performance when threading is introduced. We wish to see how effective additional threads are in increasing performance.

4.1 Throughput

We begin by examining performance of the Xeon platform using SPECWeb99, which is shown in Figure 1 for both the Apache and Flash Web servers. The five system configurations are indicated in the legend. As can be seen, performance degrades slightly when the SMP kernel is used instead of the UP kernel in the uniprocessor case. However, performance improves steadily as a second thread is added (2T), then a second processor (2P), and finally two processors each with two threads (4T). Figure 2 shows the performance exhibited by the POWER5 platform for the same experiments. Observe that performance increases in a similar fashion as is shown in Figure 1.

4.2 Speedups

While both figures show similar trends, less obvious is the *relative* increase in performance when SMT is added. This is the focus of Figure 3, which shows the improvement in performance for the two platforms. The left side of the graph shows the improvement going from 1 thread to 2 threads in the uniprocessor case (2T/1P), while the right side of the graph shows the improvement in the dual-processor case (4T/2P). Note that the performance improvement for the Xeon is significantly smaller than POWER5. For example, in the uniprocessor case with Apache, the Xeon gets a respectable 32% improvement, but POWER5 exhibits a 43% gain. The largest discrepancy is in the dual-processor scenario with Flash, where the Xeon gets only a 4% improvement when activating SMT, whereas POWER5 increases by 38%.

4.3 Static Workloads

Figures 4, 5, and 6 show the corresponding performance numbers and improvement for the two platforms using the SPECWeb99 static workloads. Note that in these graphs that the Tux Web server is included as well [1]. While the respective *numbers* are different using this workload, the *trends* are all the same: enabling SMT increases performance, for both the uniprocessor and dual-processor cases, but POWER5 gets a substantially better improvement than does the Xeon.

Having seen the impact of introducing SMT on performance, the next question is: why is SMT more effective for POWER5 than for the Xeon? We study this issue in depth in the next section.

[1] Due to a problem with dynamic content on Tux in POWER5, we were not able to have SPECWeb99 results with Tux available in time for submission. We expect to have this problem resolved shortly.

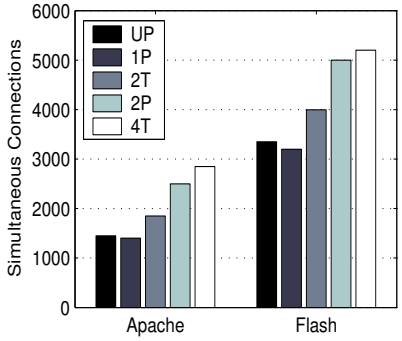

Figure 1: SPECWeb99 results on the Xeon

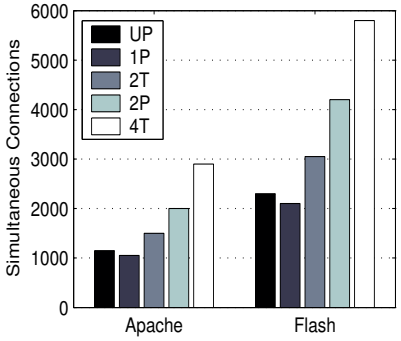

Figure 2: SPECWeb99 results on POWER5

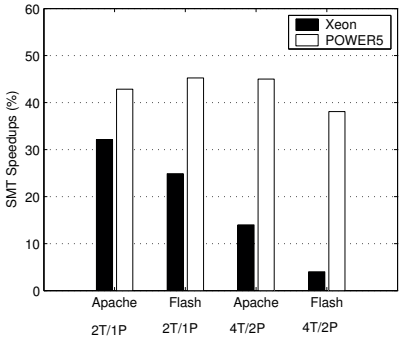

Figure 3: SMT speedups with SPECWeb99 (Xeon & POWER5)

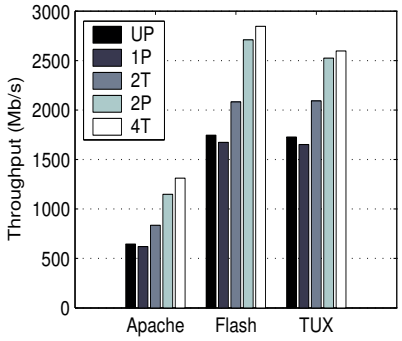

Figure 4: SPECWeb99 static results on the Xeon

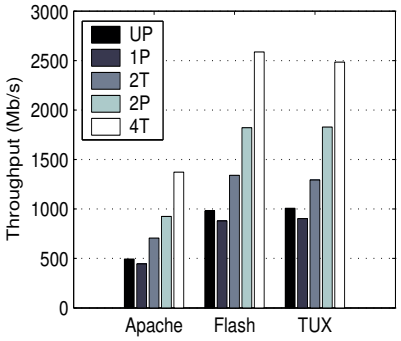

Figure 5: SPECWeb99 static results on POWER5

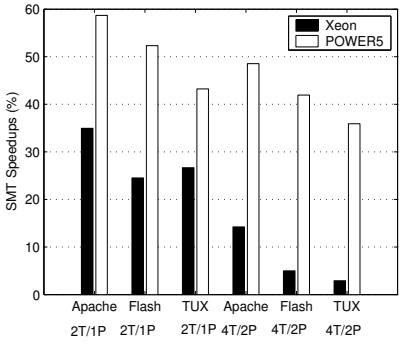

Figure 6: SMT speedups with SPECWeb99 static (Xeon & POWER5)

5 Architectural Analysis

In this section we perform an architectural analysis to understand the underlying causes of the performance observed in the previous section. We use the architectural performance counters available on each processor to measure the cycles per instruction (CPI) for each server, workload, configuration, and processor. In addition, we use the counters to to compare the behavior of the processors, examining cycle-intensive events such as cache misses, TLB misses, pipeline stalls, branch mispredicts, etc. By observing how the frequency of these events change in response to the introduction of SMT, we can determine whether and how they are sensitive to threading, and thus how SMT stresses the system.

5.1 Performance Counters

Both the Xeon and POWER5 processors offer a wide range of performance counters that measure architectural events such as cache misses, branch mispredicts, pipeline stalls, etc., in a fashion similar to Digital's (now HP) DCPI [1]. The Xeon provides about 40 categories and hundreds of events to monitor. POWER5 provides even richer information with performance counters. It has more than a hundred groups of counters with each group having 5 events. Two of the events, instructions completed and run cycles, are included in all groups. Since both the Xeon and POWER5 have speculative execution and out of order completion, monitoring and interpreting performance counters on these processors to determine the exact contribution to CPI is not straightforward.

The Xeon, for example, does not have a mechanism to measure the exact number of stalled cycles caused by a particular event. POWER5 provides some support to measure the penalty caused by an event, although it is not complete either. These events include: base completion cycles; global completion table (GCT) empty cycles caused by I-cache miss penalty; branch misprediction penalty and store stall penalty; and other stall cycles such as stalls by load/store instructions, and stalls by fixed-point or floating-point instructions. We estimate the contribution of an event to CPI based on the frequency of the event as measured by the performance counters multiplied by the cost of the events. Costs are either measured directly via lmbench [26] (e.g., cache memory, main memory, or TLB access times) or calculated based on values taken from the appropriate processor references [14, 16, 17, 22, 23, 28, 38].

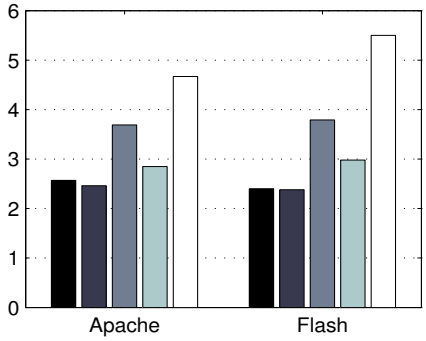

Figure 7: Measured CPI of the Xeon

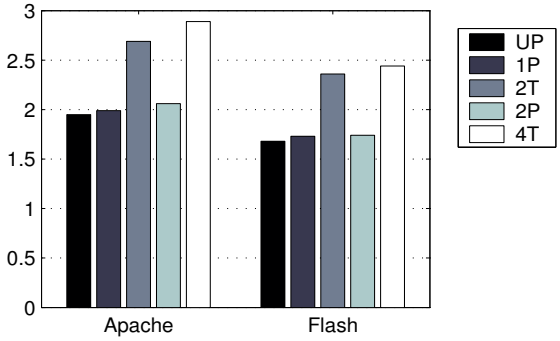

Figure 8: Measured CPI of POWER5

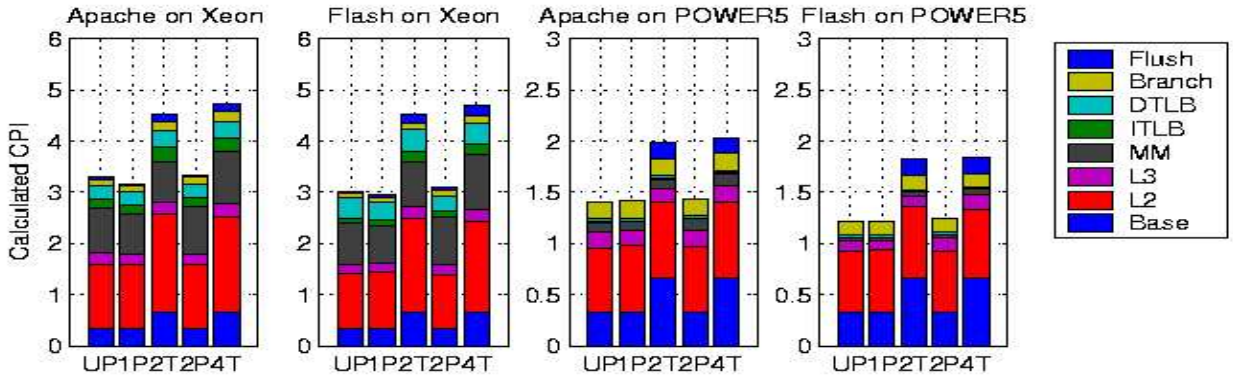

Figure 9: Calculated CPI breakdown

5.2 Cycles Per Instruction

In evaluating processor performance, perhaps the most common metric is cycles per instruction (CPI). We measure CPI by dividing the number of run cycles completed by the number of instructions retired. For the Xeon, we use the RISC-like micro-ops (μops) when measuring instructions. Our goal is to explain the changes in performance due to SMT via corresponding changes in CPI. Our definition of CPI is thus *per-thread* CPI rather than *per-processor*. For example, in an idealized case, CPI would not change when SMT is introduced, which would lead to a doubling of processor throughput.

Figure 7 shows the measured CPIs of the Xeon and Figure 8 shows the corresponding CPIs for POWER5. The key feature to observe in these graphs is the increase in CPI when SMT is enabled. For example, in the Xeon, the CPI when running Apache jumps from 2.2 in the single-threaded uniprocessor case (1P) to 3.8 in the dual-threaded case (2T), roughly a 57% increase. Similarly, the Xeon's CPI nearly doubles from 3 to 5.6 when changing from a single-threaded dual-processor scenario (2P) to the dual-threaded multiprocessor case (4T). POWER5, on the other hand, sees only a 20-30% increase in CPI across the same configurations. This shows that POWER5 is making better use of SMT.

	Xeon	POWER5
L2 fetch	18	12
L3 fetch	42	67
Main memory fetch	362	250
ITLB miss	65	28
ISLB miss	N/A	48
DTLB miss	65	28
ISLB miss	N/A	48
Branch Misprediction	20	11
Pipeline Flush	30	20

Table 4: Cost of each latency component in cycles, measured by lmbench or calculated from processor references

The next question, then, is to determine the origins of the CPI increase for each of the platforms. In order to understand the performance of hardware components as well as their impact to the overall performance, we break down CPI into its component events such as cache misses, TLB misses, branch misprediction, etc. into a calculated CPI as described in Section 5.1. The cycle costs used in these calculations are shown in Table 4. Figure 9 shows the results of this breakdown. As can be seen, the cal-

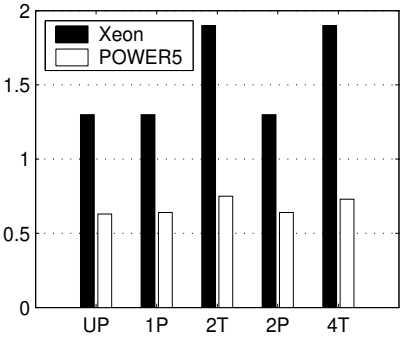

Figure 10: CPI of Apache L2 references

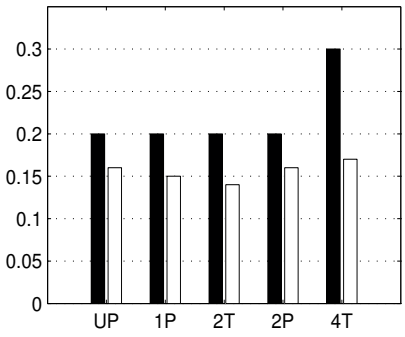

Figure 11: CPI of Apache L3 references

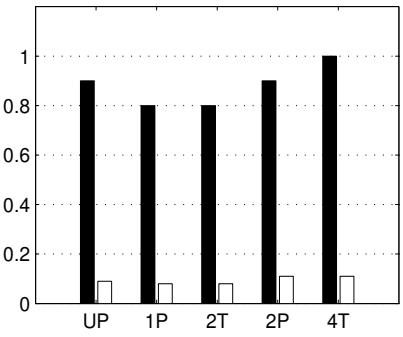

Figure 12: CPI of Apache main memory references

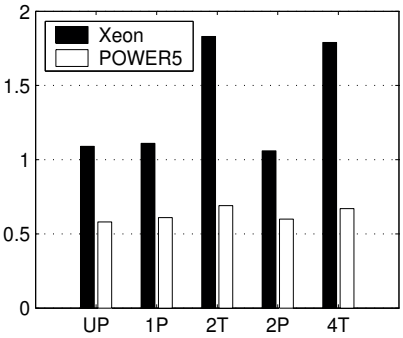

Figure 13: CPI of Flash L2 references

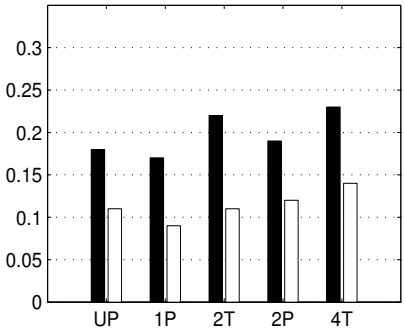

Figure 14: CPI of Flash L3 references

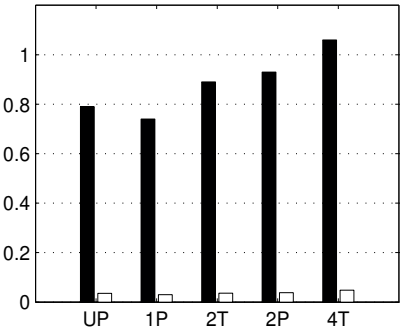

Figure 15: CPI of Flash main memory references

	Xeon		POWER5	
	2T/1P	4T/2P	2T/1P	4T/2P
Apache Measured	1.23	1.82	0.70	0.83
Apache Calculated	1.34	1.40	0.59	0.61
Flash Measured	1.42	2.52	0.63	0.70
Flash Calculated	1.71	1.91	0.61	0.62

Table 5: CPI increase per logical processor when SMT enabled

culated CPIs match well with the observed CPIs in Figure 7 and Figure 8, particularly in terms of exposing the influence of SMT. The increases in CPI per logical processor are shown in Table 5, separated by measured increase and calculated increase. The differences in most cases are narrow, and the inexactness is due to the fact that our calculated CPI comes from performance counter events rather than cycle-accurate simulation. Since the performance counters provide only limited insight into the extent of operation overlap, we cannot precisely calculate CPI. However, our measurements are quite close, especially when viewed relative to baseline CPI. For example, the largest absolute difference occurs in the 4T/2P case for Flash on the Xeon. Here, the measured increase is 2.52, while the calculated is only 1.91, a difference of .61. However, the total CPI for that case is 5.6, so the relative error is less than 11%. Given the huge speed advantages of measurement over cycle-level simulation, a maximum error of 11% is quite tolerable. In the following sections, we separate the components of the CPI increase.

5.3 Cache & Memory Subsystem

The importance of the cache memory subsystem to overall performance is well-known in computer systems. For SMT-capable processors, memory subsystem performance is even more critical due to the sharing of the caches, memory buses and main memory. Previous work [36] has shown that an L3 cache helps the Xeon in realizing better SMT benefits. The Xeon processor used in our experiments has a 1MB L3 cache, which is usually available only on Intel's multi-processor Xeon.

Both of the processors we use have three levels of cache hierarchy, and we configure the same amount of main memory size on each. However, the cache subsystems on the two processors differ in several ways. The POWER5 has much larger cache size than the Xeon, as shown in Table 2. Though access latencies of the three levels of caches are comparable in terms of cycles, main memory latency of POWER5 is about two thirds that of the Xeon. The first level instruction cache on the Xeon is called the trace cache, which stores decoded instructions

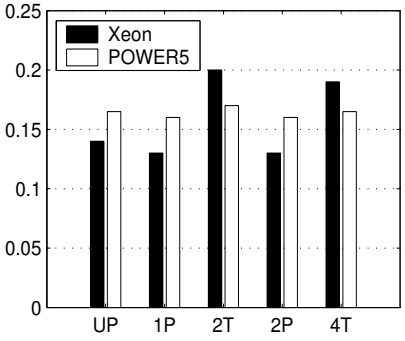

Figure 16: CPI of Apache branch misprediction

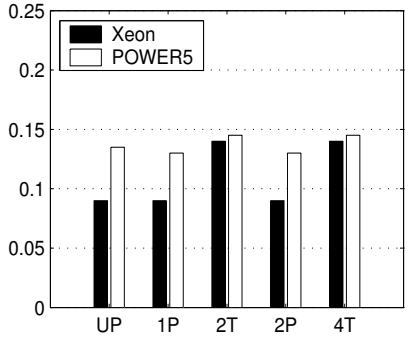

Figure 17: CPI of Flash branch misprediction

and is equivalent to a traditional L1 instruction cache. The L2 and L3 caches hold both instruction and data on the Xeon and POWER5, but are shared by the two cores on the POWER5, as it has two cores on a single chip.

We measure the number of L2, L3 and main memory accesses, calculating their overall contribution to CPI. Figures 10 through 15 show these contributions for both the Xeon and POWER5. We do not discuss L1 hit cycles here because they take 2 cycles on each of the processors and usually can be overlapped by other activities. We discuss each Figure in detail below.

Figure 10 and Figure 13 show CPI contribution attributed to fetches from the L2 cache, using Apache and Flash. Though both the Xeon and the POWER5 exhibit more L2 accesses when SMT is enabled, the POWER5 is less affected by this pressure. L2 contribution to CPI of the Xeon increases about 50% in SMT mode, while POWER5 increases only 16%. This by itself accounts for a third of the overall increase in CPI that the Xeon exhibits in Figure 7. Cache size is likely a significant factor here, since the L1 instruction cache on POWER5 is 5 times larger than on the Xeon and the L1 data cache is 4 times larger. Whether or not set associativity plays an role here can not be determined because the performance counters do not separately track conflict misses.

CPI contributed by references that hit in L3 are shown in Figure 11 and Figure 14. The contribution of L3 accesses to CPI is relatively unaffected by the introduction of SMT. The amount of CPI contributed by L3 accesses is also much less than those caused by hits to L2 and main memory. We surmise this is because of the footprint of our workloads – most of the accesses are absorbed by the L2 cache.

Figure 12 and Figure 15 present the contribution to CPI caused by accesses that reach main memory. The results are significantly different between the Xeon and POWER5. Memory references consist of about 15-20% of overall CPI on the Xeon, while it is only about 5% on POWER5. We do see some increasing contribution to CPI as SMT is introduced on the Xeon, but relatively little change on POWER5. Recall that the L3 cache size on POWER5 is 36 MB, shared by each core, but is only 1 MB on the Xeon. Moreover, it takes about 250 cycles to reach the main memory on POWER5 while the latency on the Xeon is more than 360 cycles. We thus conclude that SMT stresses the memory system more on Xeon than POWER5, most likely because of the latter's larger caches.

5.4 Branch Prediction Unit

Branches are usually a significant fraction of instructions used in server applications, and comprise about 15-17% of the instructions used by both Apache and Flash. Since both of the processors use speculative execution, a branch target needs to be predicted to execute before the branch instruction is evaluated. If a branch is mispredicted, all instructions following the branch need to be flushed from the pipeline, causing a delay. This penalty is about 20 cycles on the Xeon and 11 cycles on POWER5.

On each of the processors, the branch prediction unit is shared by the hardware threads when running with SMT enabled. Branch misprediction rates could thus suffer when SMT is enabled, and in fact we do see this on the Xeon. Apache has about 7% mispredition rate in single thread mode and 10-12% in SMT mode. However, on POWER5, both Apache and Flash have about 10% branch misprediction rate on POWER5. The difference between single thread mode and SMT mode is minimal. We measure the number of mispredicted branches and calculate their contribution to the CPI. The results are shown in Figure 16 and Figure 17. Observe that when SMT is enabled, the contribution of branch misprediction to CPI increases nearly 50%, or 0.05. However, this is a relatively small contribution to the overall CPI on the Xeon, which has an increase of 1.5 to CPI when SMT is utilized.

The different misprediction phenomenon in responding to SMT comes from the design of branch prediction unit on the two processors. The branch prediction

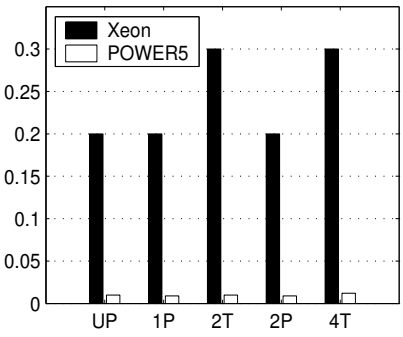

Figure 18: CPI of Apache ITLB misses

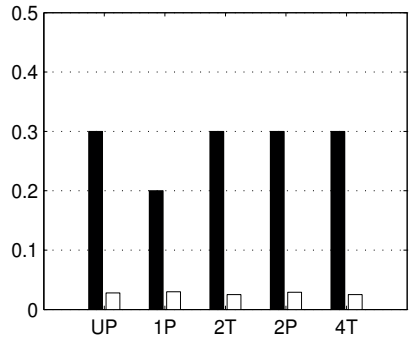

Figure 19: CPI of Apache DTLB misses

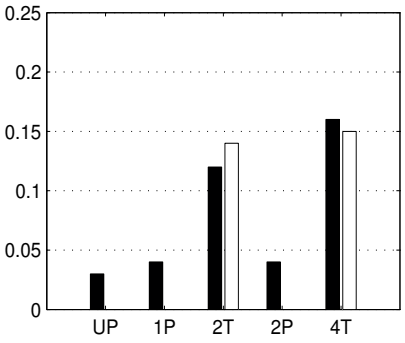

Figure 20: CPI of Apache pipeline clears

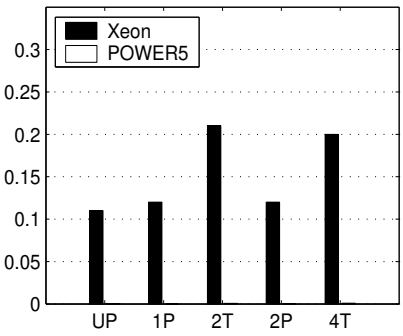

Figure 21: CPI of Flash ITLB misses

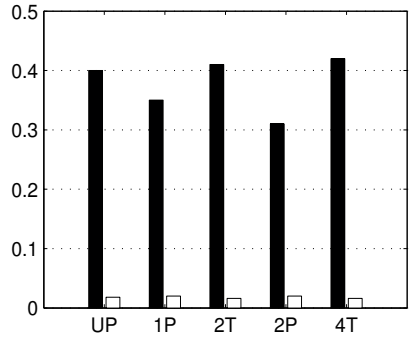

Figure 22: CPI of Flash DTLB misses

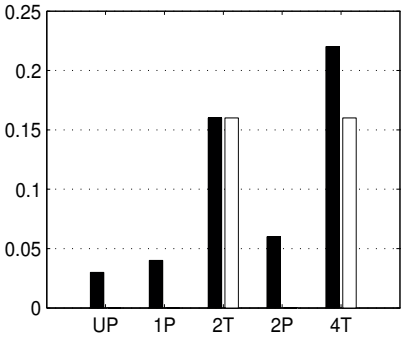

Figure 23: CPI of Flash pipeline clears

unit on the Xeon consists of a branch history table and a branch target buffer of 4K entries. However, POWER seems to only have branch history table, and a 32-entry target cache which is only used for a subset of instructions. It uses an 8-entry link stack to predict targets of subroutine returns. The target address of most branches are calculated from the instruction's address plus and offset as described by the POWER architecture. The design of POWER5 branch prediction may be less sensitive to SMT, but has a higher misprediction rate than does the Xeon. Our finding about insignificance between single thread mode and SMT mode on POWER5 is consistent with some of the previous research [23], which exercised computation-intensive workloads and compared the design of POWER4 and POWER5.

5.5 Address Translation Resources

A translation lookaside buffer (TLB) is commonly used in modern processors to translate virtual addresses to physical addresses. The Xeon has a 128-entry instruction TLB (ITLB) and a 64-entry data TLB (DTLB). POWER5 has a multi-layer address translation mechanism. The address visible to software is called the effective address. In order to access the hardware, the effective address first needs to be translated to a virtual address and then to a real address. The processor also provides a segment lookaside buffer (SLB) in addition to the TLB. Each processor core contains a unified, 1024 entry TLB. There are two 128-entry caches to facilitate the effective-to-real address translation (ERAT), one for instruction addresses and one for data addresses. For most accesses, address translation is satisfied with a hit in the ERAT cache. The SLB and TLB are utilized only if the ERAT access is a miss.

Figure 18 and Figure 21 show the cost to CPI of instruction TLB misses of the Xeon and accumulated miss penalties of TLB, SLB and ERAT on POWER5. Data TLB miss costs on the Xeon and data miss costs of TLB, SLB, ERAT on POWER5 are shown in Figure 19 and Figure 22. Note that the cost of ITLB misses increases 50% on the Xeon when SMT is introduced, whereas the cost on POWER5 is unchanged.

5.6 Pipeline Clear

Branch misprediction and interrupts are common events which may cause instructions in the pipeline be flushed, also called a pipeline clear. Section 5.4 presented the impact of pipeline clears due to branch mispredicts; here we show the effects of pipeline clears due to other factors, excluding branch mispredicts.

Figure 20 and Figure 23 illustrate the impact of this event. Both processors show a significant increase in CPI due to clears when SMT is enabled. However, the overall contribution to CPI of clears on POWER5 of clears in SMT mode is much larger than on the Xeon. When

in single-threaded mode, POWER5 shows a very small number of clears, whereas the Xeon still has a noticeable amount.

POWER5 has the unique aspect that, when running in SMT mode, pipeline clears can be caused by imbalanced threads or synch instructions. Imbalanced threads occurs when one thread occupies too many entries in the global completion table. The synch instruction orders memory operations across multiple processors and may require a long time to complete. A thread executing this instruction is very likely to be waiting for dispatching or decoding, and consequently block the other thread from proceeding. A further examination of the data reveals that most of the clears on POWER5 are caused by synchronization. Although the relative increase from the single thread to SMT is high, the overall contribution of the event to CPI is small, at most 6-8%. Most clears on the Xeon are caused by interrupts.

5.7 Microarchitecture Summary

At a high level, we can summarize our microarchitectural findings by noting that the Xeon's high clock rate puts heavy demands on its memory system, and it compensates by more aggressive speculation coupled with seemingly more graceful recovery from pipeline and other problems. Still, the effect of SMT is evident on the CPI, which almost doubles due to resource contention. With the Xeon's high clock rate and relatively distant memory, lower functional unit utilization stems from memory latency bottlenecks, rather than failure to have enough outstanding memory accesses. The Xeon otherwise shows no significant increases in other microarchitectural hazards stemming from the addition of SMT support. While some undesirable microarchitectural events do increase in frequency, their contribution to the CPI increase is minimal.

The POWER5, with a clock rate almost half that of the Xeon, achieves more work per cycle, and uses its well-provisioned cache to compensate for a cycle time that would normally make it uncompetitive on integer-heavy workloads. The use of SMT appears to negatively affect the POWER5 CPI much less than the Xeon, largely due to the higher effective bandwidth provided by the L2 and L3 caches. However, some architectural problems are evident with SMT enabled – pipeline flushes become a measurable contributor to CPI, and coupled with some functional unit contention, are the main limits to SMT's performance improvements.

6 Related Work

To the best of our knowledge, the first study of the impact of operating systems on SMT performance was performed by Redstone et al. [34], using the SMTSIM simulator [44] ported to the SimOS framework [35]. McDowell et al. [24] used the same simulator and studied memory allocation and synchronization strategies for a search engine application. These studies found that OS behavior had a significant impact that could not be easily neglected in the simulator. In comparison, the OS could be safely ignored in the computation-intensive workloads previously studied, including SPEC CINT and CFP [40, 43, 44], parallel ray-tracing applications [15], SPLASH-2 benchmarks [21], MPEG-2 decompression [9, 37], and other scientific application workloads [19, 39].

The difference between the SMT simulations and actual hardware was earlier explored by our previous work [36]. In that paper we examined the performance of various Web server packages across different models of the Xeon processor. The conclusion we drew from that work was that the latency of memory access was a primary barrier to low SMT performance. In this work, we have examined the differences in performance between two different processor families, the Xeon and the POWER5. This approach not only provides a broader range of parameters than just processors in a single family, but it also highlights the impact of different architectural and implementation choices.

Other researchers have also been exploring delivered SMT performance, but with a general focus on compute-intensive benchmarks. Tuck and Tullsen [42] 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 [7] 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. [9] also only evaluated performance of SMT and SMP individually.

Other work has focused on trying to improve the performance of SMT processors, often stemming from observations of microarchitectural resource conflicts. Several groups have proposed special schedulers for SMT – Snavely et al. [39, 40] use a symbiotic approach, Bulpin and Pratt [8] try to manage resource interference, and Fedorova et al. [13] try to partition caches (although for a CMP instead of an SMT). Raasch and Reinhardt [33] also study the impact of resource partitioning on SMT processors.

Performance analysis using hardware provided event counters has been an effective approach in previous studies. Bhandarkar et al. [4] and Keeton et al. [18] 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.

7 Conclusions and Future Work

In this paper, we have examined both the macroscopic and microarchitectural performance of server workloads on two very different SMT-enabled processors. By comparing behavior across two different processor families, we can get a broader perspective than previous work that examined SMT performance within a single processor family. Our overall results are fairly surprising – although the Xeon and the POWER5 have very different design philosophies, different architectural styles, different target markets, and different implementation focuses, we have shown that their respective performance gains from the use of SMT are largely tied to their memory organizations. The POWER5, with its slower cycle time and more aggressive memory hierarchy, has a much lower effective memory latency, and sees a more predictable and much higher relative performance gain than the Xeon.

All of the other architectural aspects, such as pipeline length, flushes, stalls, TLB misses, etc., vary widely between the two systems, and reflect their different implementation choices, but ultimately, all of these effects contribute less to the CPI than the memory effects. The lack of significant problems arising from microarchitectural choices may be the result of careful implementations in each processor, or it may be due to the large difference between processor speeds and memory today. In any case, even in these areas, we find microarchitectural choices that we believe can be improved to better support SMT on the workloads we consider. To the best of our knowledge, this work is the first to identify some of these opportunities, especially on the POWER5 processor.

In terms of our avenues for future work, we expect that the evolution of the POWER series will provide more opportunities to study some of the issues we have raised – in particular, the POWER5 is now shipping at a much higher clock speed (2.5GHz) than our test machine, so main memory latency may be impacted. Likewise, future generations of the POWER processor are rumored to meet or exceed the clock frequency of our test Xeon processor, so it will be interest to examine if its larger L3 cache is able to offset the higher rate of memory accesses one can expect.

In terms of future Xeon processors, the future research opportunities are less clear, since the current trend appears to be adopting multi-core processors while dropping SMT support. While this choice is, in all likelihood, a side-effect of using the older and lower-power Pentium-III/Pentium-M architecture as the basis for the multicore chips, it is not clear if SMT will reappear in the Xeon roadmap in the near future. While SMT did provide some performance boost for the Xeon, and may provide an attractive return on the investment in silicon space, the literature on the NetBurst (P4) architecture mentions that it was designed to support SMT. Adding it to the less-speculative P3 architecture may yield more pronounced penalties for pipeline problems and misspeculation.

References

[1] J.-A. M. Anderson, L. M. Berc, J. Dean, S. Ghemawat, M. R. Henzinger, S.-T. Leung, R. L. Sites, M. T. Vandevoorde, C. A. Waldspurger, and W. E. Weihl. Continuous profiling: Where have all the cycles gone? In *Proc. of the 16th ACM Symp. on Operating System Principles*, pages 1–14, 1997.

[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)*, pages 288–298, Feb. 1997.

[5] S. Blackburn, P. Cheng, and K. McKinley. Myths and realities: The performance impact of garbage collection. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, June 2004.

[6] J. M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S. R. Kunkel. A multithreaded powerpc processor for commercial servers. *IBM Journal of Research and Development*, 44(6):885–898, 2000.

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

[8] J. R. Bulpin and I. A. Pratt. Hyper-threading aware process scheduling heuristics. In *USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[9] 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.

[10] Christian Stork. Exploring the Tera MTA by example.

[11] S. Eggers, J. Emer, H. Levy, J. Lo, R. Stamm, and D. Tullsen. Simultaneous multithreading: A platform for next-generation processors. *IEEE Micro*, pages 12–18, Sept. 1997.

[12] F. Eskesen, M. Hack, T. Kimbrel, M. Squillante, R. Eickemeyer, and S. Kunkelz. Performance analysis of simultaneous multithreading in a PowerPC -based processor. In *Workshop on Duplicating, Deconstructing, and Debunking (WDDD04)*, June 2004.

[13] A. Fedorova, M. Seltzer, C. Small, and D. Nussbaum. Performance of multithreaded chip multiprocessor and implications for operating system design. In *USENIX Annual Technical Conference*, Anaheim, CA, April 2005.

[14] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the Pentium 4 processor. *Intel Technology Journal*, Feb. 2001.

[15] 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 annual International Symp. on Computer Architecture*, pages 136–145. ACM Press, 1992.

[16] Intel Corporation. Intel Pentium 4 and Intel Xeon Processor optimization reference manual. http://developer .intel.com/design/pentium4/manuals/index_new.htm.

[17] R. Kalla, B. Sinharoy, and J. M. Tendler. IBM Power5 chip: A dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, March/April 2004.

[18] 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 Annual International Symp. on Computer Architecture*, pages 15–26, 1998.

[19] H. Kwak, B. Lee, A. Hurson, S.-H. Yoon, and W.-J. Hahn. Effects of multithreading on cache performance. *IEEE Trans. Computers*, 48(2):176–184, 1999.

[20] 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 Annual International Symp. on Computer Architecture*, pages 39–50, 1998.

[21] 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):322–354, 1997.

[22] 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):4–15, Feb. 2002.

[23] H. M. Mathis, A. Mericas, J. McCalpin, R. Eickemeyer, and S. Kunkel. Characterizaton of simultaneous multithreading (SMT) efficiency in POWER5. *IBM Journal of Research and Development*, 49(4/5), July/September 2005.

[24] L. McDowell, S. Eggers, and S. Gribble. Improving server software support for simultaneous multithreaded processors. In *PPoPP 2003*, pages 37–48, San Diego, CA, June 2003.

[25] C. McNairy and R. Bhatia. Montecito: A dual-core, dual-thread itanium processor. *IEEE Micro*, 26(2):10–20, March/April 2005.

[26] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference*, pages 279–294, San Diego, CA, June 1996.

[27] B. Olszewski and O. Herescu. Performance workloads in a hardware multi threaded environment. In *Fifth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, Boston, MA, February 2002.

[28] B. Olszewski, O. Herescu, and H. Hua. Performance workloads characterization on POWER5 with simultaneous multi threading support. In *Eighth Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)*, San Francisco, CA, February 2005.

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

[30] V. Pai, P. Druschel, and W. Zwaenepoel. Flash: An efficient and portable web server. In *USENIX Annual Technical Conference*, pages 199–212, Monterey, CA, June 1999.

[31] K. O. Poonacha Kongetira, Kathirgamar Aingaran. Niagara: A 32-way multithreaded sparc processor. *IEEE Micro*, 26(2):21–29, March/April 2005.

[32] R. E. Hiromoto, O. M. Lubeck, and J. Moore. Experiences with the Denelcor HEP. In *Parallel Computing*, pages 197–206, 1984.

[33] S. Raasch and S. Reinhardt. The impact of resource partitioning on SMT processors. In *PACT 12*, New Orleans, Louisiana, Sept. 2003.

[34] J. Redstone, S. Eggers, and H. Levy. An analysis of operating system behavior on a simultaneous multithreaded architecture. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 245–256, 2000.

[35] 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):34–43, Winter 1995.

[36] Y. Ruan, V. Pai, E. Nahum, and J. Tracey. Evaluating the impact of simultaneous multithreading on network servers using real hardware. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, Banff, Canada, June 2005.

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

[38] B. Sinaroy, R. N. Kalla, J. M. Tendler, R. Eickemeyer, and J.B.Joyner. POWER5 system microarchitecture. *IBM Journal of Research and Development*, 49(4/5), July/September 2005.

[39] A. Snavely and D. Tullsen. Symbiotic job scheduling for a simultaneous multithreaded processor. In *Proc. of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 234–244. ACM Press, 2000.

[40] A. Snavely, D. Tullsen, and G. Voelker. Symbiotic jobscheduling with priorities for a simultaneous multithreading processor. In *Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 66–76, June 2002.

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

[42] N. Tuck and D. Tullsen. Initial observations of a simultaneous multithreading processor. In *PACT 12*, Sept. 2003.

[43] 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 23rd Annual International Symp. on Computer Architecture*, pages 191–202, 1996.

[44] D. Tullsen, S. Eggers, and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proc. of the 22nd Annual International Symp. on Computer Architecture*, 1995.