Do Commodity SMT Processors Need More OS Research?

Yaoping Ruan, Vivek S. Pai, Erich Nahum[†], and John Tracey[†]
Department of Computer Science, Princeton University
{yruan,vivek}@cs.princeton.edu
[†] *IBM T.J.Watson Research Center*
{nahum,traceyj}@us.ibm.com

Abstract

The availability of Simultaneous Multithreading (SMT) in commodity processors such as the Pentium 4 (P4) has raised interest among OS researchers. While earlier simulation studies of SMT suggested exciting performance potential, observed improvement on the P4 has been much more restrained, raising the hope that OS research can help bridge the gap. We argue that OS research for current commodity Simultaneous Multithreading (SMT) processors is unlikely to yield significant benefits. In general, we find that SMT processor simulations were extremely optimistic about cache and memory performance characteristics, while overlooking the OS overheads of SMT kernels versus uniprocessor kernels. Using measurement and analysis on actual hardware, we find that little opportunity exists for realistic performance gains on commodity SMT beyond what is currently achieved.

1 Introduction

Simultaneous Multithreading (SMT), a technique for improving processor performance, has become widely available through its incorporation in the Intel Pentium 4 (P4) series of processors. While at first restricted to only the high-end Xeon series, SMT is now also available in the commodity-oriented non-Xeon P4 processors. As a gross simplification, SMT processors utilize additional hardware threads to utilize otherwise idle functional units. These additional hardware contexts are generally presented to the operating system as additional logical processors. As a result, the P4 has made many people first-time owners of (logically) dual-processor systems.

Not surprisingly, the advent of real hardware has moved research on SMT beyond the simulation-based studies [7, 9, 14]. Much of the early work has focused on evaluation [2, 10, 12, 15] and scheduling optimization [1, 3, 8]. In general, the delivered benefits of SMT on the P4 have not matched the high expectations from the simulation studies, leaving OS researchers with a possible opportunity to narrow the gap in performance gains.

What is surprising is not that a gap exists between the simulation studies and the actual processor, but rather that the magnitude of the gap is as large as it is. In general, the simulations used 4-8 threads, and often had the first few threads seeing additional performance gains of 70-100% [13]. In comparison, the P4 has only two threads, and the observed performance gain of the second thread is generally no more than 20-30%, and often much lower.

We believe that the SMT performance of the P4 is not due to any weakness of the OS, but is instead what can be expected from SMT on commodity processors. Furthermore, we believe that more OS research to support these processors is not necessary, and will have marginal benefit. We do not arrive at this conclusion arbitrarily – it is motivated by several observations: our own work analyzing why Web server performance on the P4 differs from simulations, P4 SMT scheduling analysis using measurements from other researchers, and examination of various scenarios to make commodity processors more SMT-friendly. The observations that lead us to these conclusion are discussed in the rest of this paper, but can be summarized as follows:

Cache miss rates dominate performance – The cache structure and timings of the P4 are sufficiently different from the simulations so that the cache misses are the dominant bottleneck in the workloads we test. The extremely generous cache models used in the simulations are the main reason for the performance difference.

# CPUs	1		2		
Kernel	SMP	UP	SMP		
SMT	✗	✗	✓	✗	✓
Ap 2GHz	480	554	636	880	1016
Ap 3GHz	635	719	805	1047	1091
Ap 3G/L3	759	873	978	1297	1476
Fl 2GHz	1224	1481	1589	2082	2186
Fl 3GHz	1604	1821	1993	2352	2265
Fl 3G/L3	1796	2190	2260	2596	2685
Hb 2GHz	454	498	479	629	585
Hb 3GHz	583	609	603	745	629
Hb 3G/L3	650	624	654	797	727

Table 1: Web server throughput in Mbps of the Apache (Ap), Flash (Fl), and Haboob (Hb) Web servers on three Xeon models with Hyper-Threading. 3G/L3 is 3GHz with 1MB L3 cache.

Memory bottlenecks do not improve – The CPU is so much faster than memory that all of the memory-related components are nearly fully utilized. Adding a second thread from the same application only stresses the same resources.

Synergy is virtually nonexistent – If parallel threads do provide some synergy by prefetching code or data for each other, it is dominated by cache capacity issues. Thus performance losses due to contention may surpass benefits from the synergy.

Simple scheduling policies suffice – When trying to schedule two different programs to avoid having shared bottlenecks, very simple scheduling policies perform only moderately worse than an idealized optimal scheduler (20% vs. 23%).

Historically, these problems have worsened – Using history as a guide, all of the issues that prevent good SMT performance on commodity processors are likely to get worse going forward.

2 Understanding SMT Performance

In this section, we examine the performance of the P4 SMT on Linux. We first examine the performance of Web workloads that have been used in simulation studies and discuss their measured bottlenecks. We also examine the scheduling possibilities for compute-intensive workloads, using previously-published data. Finally, we compare why the measured gains are very different from the simulation studies.

	versus 1P-SMP			versus 1P-UP		
	2T	2P	rltv	2T	2P	rltv
Ap 2GHz	32	83	39	15	59	25
Ap 3GHz	27	65	41	12	46	26
Ap 3G/L3	29	71	41	12	49	25
Fl 2GHz	30	70	42	7	41	18
Fl 3GHz	24	47	52	9	29	32
Fl 3G/L3	26	44	58	3	18	17

Table 2: Relative throughput gains (in %) – columns are percentage gains of SMT-enabled uniprocessor (2T) and 2 processors (2P) versus uniprocessor base case with SMP kernel (1P-SMP) and uniprocessor kernel (1P-UP). Rltv column indicates what percentage of 2P gain was achieved by SMT. For example, at 2GHz with an SMP kernel, enabling SMT increases Apache's performance by 32%. Adding a second processor instead of using SMT increases performance by 83%, so using SMT captures 39% of the gain of a second processor.

2.1 Web Server Performance

To compare our results with earlier simulation studies [7, 9], we run tests of the Apache Web Server (and others) on the SPECWeb96 benchmark, with the results shown in Table 1. Although SPECWeb99 is more recent, we use SPECWeb96 because it was used in the simulation studies. We use three versions of the Pentium 4 Xeon: a 3GHz processor with an 1MB L3 cache, and versions without the L3 cache running at 2 GHz and 3 GHz. All processors are tested on the same motherboard with 4GB memory and 4 Gigabit Ethernet adapters. The data set size of the workload is 500MB. For the non-SMT uniprocessor case, we run both a uniprocessor kernel as well as an SMP-enabled kernel. The kernel is Linux 2.6.8.1, and the SMP kernel has SMT optimizations.

Apache gains 27-32% with SMT enabled if the SMP kernel overhead is ignored, but these gains drop to 12-15% when comparing with a uniprocessor kernel, as shown in Table 2. We can also compare the relative gain of SMT versus using a second physical processor – this value is roughly 40% in the SMP base case, and 25% in the uniprocessor base case. While running an SMP kernel on a non-SMT uniprocessor is clearly a bad idea and leads to a 15% performance loss, simulation studies use only SMP kernels as their base case. These results suggest that multicore chips are perhaps more

	L1-I (%)	L1-D (%)	L2 (%)	Bus (%)	CPI	
					All	L1+2
1P	10.5	4.6	4.8	8.7	7.7	6.9
1P/SMT	17.0	5.7	3.8	13.3	10.9	9.3
2P	10.8	4.6	5.2	15.4	8.1	7.2
2P/SMT	17.6	5.7	4.0	18.7	11.0	10.0

Table 3: L1, L2 miss rate, bus utilization, and calculated Cycles per Instruction (CPI) per thread of the Apache server on the 3GHz Xeon without L3, using an identical SMP kernel, on uniprocessor system (1P) and dual-processor system (2P). The other two processors show similar miss rates but lower bus utilization.

promising than SMT on this workload. If SMT can only deliver one-quarter the performance of a second processor, using multiple cores may be a better use of the ever-increasing number of transistors.

Using the CPU hardware performance counters, we find that SMT-enabled systems present much higher cache pressure and bus utilization than their non-SMT counterparts on these workloads. The results for Apache are shown in Table 3, and show that L1 Icache and Dcache miss rates jump noticeably when SMT is enabled. The drop in L2 cache miss rates is misleading – the absolute number of misses has increased, but the capacity problems in the L1 cache cause more hits in the L2 cache.

This effect is also visible in the bus utilization and CPI values, which all increase when SMT is enabled. Even if we normalize bus utilization versus throughput, the effect is still evident – the utilization per Gbps in % is 13.7 (1P), 16.5 (1P/SMT), 14.7 (2P), and 17.1 (2P/SMT). The fact that the 2P bus utilization is higher than the 1P/SMT case indicates that any synergistic effects are outweighed by the capacity pressure. A more detailed analysis of the microarchitectural events indicates that cache reference time contributes more than 80% of the total CPI in the SMT cases.

Cache pressure bottlenecks are most noticeable in commodity multiprocessor systems, because their shared memory buses saturate faster. A comparison of Apache's 2P/SMT with the 2P numbers in Table 1 demonstrates this problem. The 2GHz and 3GHz/L3 cases show a 15% speedup with SMT, but the non-L3 3GHz case only sees 5%. Not only does the 3GHz processor have "farther" memory than the 2GHz processor as measured in CPU cycles, but it also lacks the L3 cache that could reduce memory traffic. For Flash, the performance actually drops for the 2P/SMT 3GHz scenario.

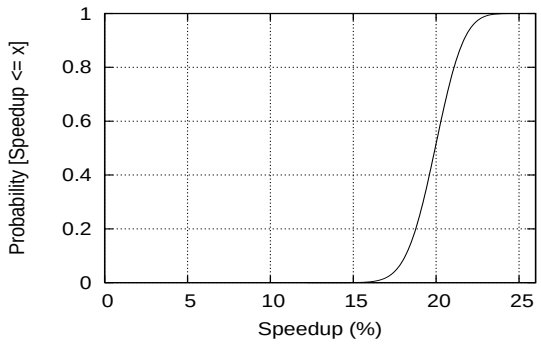

Figure 1: Speedup CDF of SPEC CPU2000 scheduling: Min is 12%, Max is 26%, and Mean is 20%.

2.2 Scheduling

One other area of interest for SMT systems is on CPU-intensive workloads, particularly in multiprogrammed systems. In particular, these programs may be more cache friendly, and may have very different resource utilization characteristics, so running two different CPU-intensive programs can potentially improve processor utilization. However, even this attempt of pairing mutually-beneficial programs has its caveats, and the associated problems have led a number of researchers to explore optimizing the scheduler for SMT [3, 11].

Rather than recreate this research, we use previously-published results but analyze them differently. In particular, two groups have examined the pairwise interference/benefit between the 26 programs in the SPEC CPU2000 [2, 12] benchmark suite. We use these as input to our analysis, which basically asks the following question: if all 26 program pairs ran for the same time, and you had to schedule them for the most benefit, how would you do it? Since the possible permutations are far too large to analyze, we use randomized schedules to simplify the analysis.

We create schedules by randomly pairing program and calculating the schedule's total runtime using the pairwise interference measurements. By repeating this process for a large number of trials, we can determine the range of scheduling benefit. The CDF of 10 million trials, shown in Figure 1, indicates that the range of resulting speedups is fairly narrow, with a mean of 20% and an absolute max-

Workloads	Speedup	
	Simulation	Measurement
SPEC CPU	❶ 80% (2T) / 150% (8T)	❷ 20% (2T)
Apache + SPEC Web	❸ 400% (8T)	❹ 16 - 28% (2T) ❺ 5 - 15% (2T)
L1 size	32 - 128KB	8 - 12KB
L2 size	256KB - 16MB	256 - 512KB
Mem cycles	62 - 90	225 - 344

Table 4: SMT speedups and hardware threads. ❶ uses the SPEC92 benchmark suite on a system derived from Alpha 21164 [13]. ❷ is from [12] using CPU2000 on an 2.5GHz P4 system. ❸ comes from [9] using a system derived from Alpha 21264. Both ❹ [6] and ❺ [10] use P4 processors.

imum of 26%. A more typical high-end schedule achieves 23%.

So, even if we have perfect offline information, the ideal scheduler achieves only 3%-6% more speedup than a random scheduler. One may argue that a random scheduler could perform as poorly as only a 12% speedup. While true, the remedy is also simple. The scheduler could periodically re-randomize the pairings, bringing all schedules closer to the median as the number of randomizations increases. This scheduling policy is extremely simple, and performs almost as well as an ideal scheduler. For realistic schedulers, which will need to collect the pairwise data at runtime, the gap may be even narrower.

While we use randomized analysis, this argument is not about the law of large numbers – we assume that the ideal scheduler can achieve the best schedule, or something close to it. What we are merely observing is that given enough programs and the pairwise characteristics of the P4, the average speedup is simply not that bad. If people run very few simultaneous programs, then scheduling matters even less – there are very few choices for the scheduler to make. Some degenerate cases are possible, where there may be only two programs in the system, and they exhibit slowdown when run together. However, these cases are rare among the SPEC CPU2000 programs, and the ability to detect this scenario does not imply that a more complicated scheduler is needed.

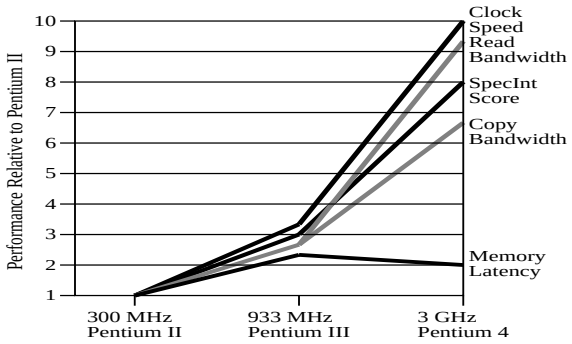

Figure 2: Scaling trends in the Intel x86 family

3 SMT History & Future

To understand the potential for commodity SMT processors, it is useful not only to compare measurements with the simulations, but also to understand the trends that are likely to shape future processors. A summary of various studies and measurements is shown in Table 4.

The hardware differences between the simulated and measured systems are dramatic, and these choices influence the speedups obtained. The main memory latencies suggest that the simulated processors have roughly a 750 MHz clock, which is very slow for modern processors. While the simulated L1 cache size is much larger than the P4's, it is comparable to what AMD is currently shipping, although the AMD processors are only available at speeds up to 2.6 GHz. The simulated L2 cache sizes are not available in any processor, although the (non-commodity) POWER5 does have a 36MB L3 cache. Not only is this L3 cache much more expensive to access in CPU cycles than the simulated L2, but the POWER5 is more than twice the clock rate of the simulated processor.

In short, the simulations use a much slower processor coupled with an unrealistically aggressive memory hierarchy. Given that we observed memory hierarchy pressure was causing CPU idleness and increasing CPI, the simulated processors would undoubtedly demonstrate much better speedup than any processor available today. At issue is whether the system balance of the simulated processors is likely to occur in the future.

A brief summary of Intel processor scaling measurements, shown in Figure 2, suggests that commodity processors are likely to move further away from the simulated processors over time. In three

processor generations, clock speed has increased 10-fold, but memory copy bandwidth has only increased 6.5-fold, and latency is barely improving in absolute terms. Latency is actually getting much worse in relative terms, because the 2-fold decrease in latency translates to a 5-fold increase in the number of clock cycles needed to access main memory. Even if the pace of CPU clock rate improvements slows, the memory latencies would still have to improve dramatically to match the simulations.

Other processors, not targeted at the commodity market, may fare better with SMT, since they have to target codes other than the cache-friendly SPECInt-like programs that benefit highly from high clock rates. In absolute terms, the IBM POWER5 [5] (which also features SMT) has a better cache hierarchy than the P4, and in relative terms, its main memory will appear closer due to its lower clock rate. The upcoming Sun "Niagara" processors [16], which are chip multithreaded (CMT) may also have different memory hierarchy behavior than the P4. Simulations suggest that their shared L2 caches may benefit from capacity management [4].

4 Conclusion

We have shown that the current performance of commodity SMT is not an aberration, but rather the result of a smaller and slower memory hierarchy than modeled by simulations. The server-style applications we have tested look unlikely to benefit from any SMT-specific optimizations as long as they continue to be bottlenecked on the cache and memory system. For compute-intensive applications, an analysis of randomized scheduling suggests that it will perform within a few percent of an ideal scheduler, without requiring any *a priori* knowledge or extra mechanism.

Our conclusion from these investigations is that commodity SMT processors do not require any significant amount of OS research, since their bottlenecks are mostly hardware-related. Given the trends in hardware performance, this situation is likely to hold true for the foreseeable future. The only significant opportunity we uncovered was not related directly to SMT, but rather to the overheads of SMP kernels in general. These overheads tended to reduce SMT performance significantly, and may be addressable. Their impact appears to be larger than any gains achievable from SMT-specific scheduling.

References

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

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

[3] J. R. Bulpin and I. A. Pratt. Hyper-threading aware process scheduling heuristics. In *USENIX 2005 Annual Tech, To appear*, Anaheim, CA, April 2005.

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

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

[6] 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.

[7] 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.

[8] J. Nakajima and V. Pallipadi. Enhancements for hyper-threading technology in the operating system: Seeking the optimal scheduling. In *Proc. of the 2nd Workshop on Industrial Experiences with Systems Software (WIESS'02)*, Boston, MA, Dec. 2002.

[9] J. Redstone, S. Eggers, and H. Levy. An analysis of operating system behavior on a simultaneous multithreaded architecture. In *ASPLOS 2000*, pages 245–256, 2000.

[10] Y. Ruan, V. Pai, E. Nahum, and J. Tracey. Evaluating the impact of simultaneous multithreading on network servers using real hardware. In *Sigmetrics 2005, To appear*, 2005.

[11] A. Snavely and D. Tullsen. Symbiotic job scheduling for a simultaneous multithreaded processor. In *ASPLOS 2000*, pages 234–244. ACM Press, 2000.

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

[13] 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 *ISCA 23*, pages 191–202, 1996.

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

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

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