Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 07-026

Performance and power comparison of Thread Level Speculation in SMT and CMP architectures

Venkatesan Packirisamy, Antonia Zhai, Wei-chung Hsu, and
Pen-chung Yew

October 30, 2007

Performance and power comparison of Thread Level Speculation in SMT and CMP architectures

Venkatesan Packirisamy, Antonia Zhai, Wei-Chung Hsu
and Pen-Chung Yew
email:{packve,zhai,hsu,yew}@cs.umn.edu
Department of Computer Science,
University of Minnesota, Minneapolis.

Abstract

As technology advances, microprocessors that support multiple threads of execution on a single chip are becoming increasingly common. Improving the performance of general purpose applications by extracting parallel threads is extremely difficult, due to the complex control flow and ambiguous data dependences that are inherent to these applications. Thread-Level Speculation (TLS) enables speculative parallel execution of potentially dependent threads, and ensures correct execution by providing hardware support to detect data dependence violations and to recover from speculation failures.

TLS can be supported on a variety of architectures, among them are Chip MultiProcessors (CMP) and Simultaneous MultiThreading (SMT). While there have been numerous papers comparing the performance and power efficiency of SMT and CMP processors under various workloads, relatively little has been done to compare them under the context of TLS. While CMPs utilize smaller and more power-efficient cores, resource sharing and constructive interference between speculative and non-speculative threads can potentially make SMT more power efficient. Thus, this paper aims to fill this void by extending a CMP and a SMT processor to support TLS, and evaluating the performance and power efficiency of the resulting systems with speculative parallel threads extracted for the SPEC2000 benchmark suite. Both SMT and CMP processors have a large variety of configurations, we choose to conduct our study on two architectures with equal die area and the same clock frequency. Our results show that a SMT processor that supports four speculative threads outperforms a CMP processor that supports the same

number of threads, uses the same die area and operates at the same clock frequency by 23% while consuming only 8% more power on selected SPEC2000 benchmarks. In terms of energy-delay product, the same SMT processor is approximately 10% more efficient than the CMP processor.

1 Introduction

As technology advances, microprocessors that support multiple threads of execution [2, 14, 36, 18, 15] on a single chip are becoming increasingly common. However, extracting independent threads for parallel execution is an extremely challenging task. Thread-Level Speculation (TLS) facilitates thread extraction by allowing the programmer/compiler to create threads that are potential dependent. The underlying hardware support ensures correct execution by detecting and recovering from possible data dependence violations. While a large number of proposals have discussed efficient hardware support for TLS [20, 1, 11, 7, 9, 33, 12, 13, 21, 27, 28, 34, 37], as well as hardware and software optimizations for improving TLS performance, relatively little has been down to evaluate the power efficiency of such execution paradigm [31].

Simultaneous Multithreading (SMT) and Chip Multithreading (CMP) are two approaches to support multiple threads of execution on a single chip. In SMT, instructions from different threads are simultaneously executed in the same pipeline, while in CMP, instruction from different threads are executed on separate cores. Though CMP is the commonly used architecture for TLS in previous proposals, SMT can also be extended to support TLS [30, 29]. The power and performance characteristics of both the SMT and the CMP architectures have been studied extensively under different workloads: while Lo *et. al* [24] show that SMT achieves better speedup for explicitly parallel workloads (from SPLASH-2), Kaxiras *et. al* [19] conclude that SMT is more power efficient. Sasanka *et al* show that CMP is more power efficient for multimedia workload and Burns *et al* [6] show CMP to have higher throughput. However, it is difficult, if not impossible, to determine which architecture is more efficient to support TLS based on previous work, since the characteristics of TLS workload is fundamentally different from those of multi-threaded workloads:

Efficiency of cores: CMPs are able to utilize smaller and more power efficient cores, while SMT must utilize large and complex cores that are less power efficient.

Available parallelism vary during execution: Unlike multi-threaded workloads, the amount of parallelism that can be exploited by TLS may vary during execution of a program. In particular, different loops can

have different amounts of parallelism; for significant portion of execution, efficient speculation threads cannot be extracted. When available speculative threads are insufficient for utilizing all hardware threads, some of the cores will idle in a CMP-based architectures, while resources can be dynamically reallocated to exploit instruction level parallelism.

Resource competition and sharing between speculative and non-speculative threads: In SMT, the non-speculative thread competes and shares resources with speculative threads. Such competition can degrade the performance of the non-speculative thread. On the other hand, resource sharing can also benefit the performance of TLS. E.g., a speculative thread fetches data into the cache during it execution, even if the speculative thread is eventually squashed, the data it brought into the cache can potentially be used by the non-speculative thread or other speculative threads. In a CMP architecture, threads only shared data that are located in the L2 cache, while in a SMT architecture, data in the L1 can be shared as well.

Power consumption due to speculation failure: When data dependence is violate, the thread that contains the consumer of the dependence must be re-executed. The number of re-execution may differ depends on the underlying hardware. We have observed that there are more false violations in SMT. This effect can increase the power consumption of SMT.

To conduct a detailed study comparing the two architectures, and to understand the relative merits of each architecture, we must identify two architectures with the same *cost*. In this paper, we choose to compare two architectures with the same die area. We use detailed area estimation tools [26] to decide which configurations to use. To the best of our knowledge, we are unaware of any previous work that compares the SMT and the CMP architectures under the context of TLS. Also our paper presents a detailed study of power behavior of TLS which so far has not been well understood in comparison to the performance aspect of TLS.

The rest of the paper is organized as follows: in Section 2, we describe the differences in TLS execution patterns for CMP and SMT; in Section 3, we specify the architectural parameter of equal area CMP and SMT processors; in Section 4, we describe our evaluation framework; in Section 5 we present a quantitative evaluation of the performance and power efficiency of CMP and SMT under the context of TLS; and finally in Section 6, we present our conclusions.

1.1 Related work

In this section, we discuss two sets of related works—previous work that evaluates the power efficiency of microprocessors that support TLS; as well as previous work that compares the performance and power-efficiency of CMP and SMT.

While a large amount of work has been devotes the study the performance of TLS [20, 1, 11, 7, 9, 33, 12, 13, 21, 27, 28, 34, 37, 3, 34, 43, 44, 8, 17, 39, 42, 23], relatively little has been devoted to evaluate the power efficiency of TLS. Renau *et. al* [31] compared a wide-issue superscalar processor with an equal-area CMP processor that supports speculative threads and found that the CMP processor is more power efficient. In this paper, our main contribution is a detailed comparison of the performance and power efficiency of TLS support on SMT and CMP processors. Our study also includes a detailed evaluation of chip area for the different chip configurations.

Numerous studies have compared the SMT and CMP performance and power efficiency under different workloads. Lo *et. al* [24], evaluate the performance of SMT processors against that of CMP processors on parallel programs and find that SMT processors outperform CMP processors. Kaxiras *et. al* [19] compare the two architectures for mobile workloads and found that SMT is more efficient. Li *et. al* [22] compare equal-area configurations of SMT and CMP for multi-programmed workloads and find that SMT is more efficient for memory bound applications while CMP is more efficient for computational intensive applications.

Sasanka et al [32] compared these two architectures in terms of energy efficiency for multimedia applications. They argue that to compare power of the two architectures, their performance must be maintained the same. So, they considered different configurations with equal performance for SMT and CMP and found that CMP is more power efficient, but consumes more chip area. In our paper, we fix the chip area to be the same and compare SMT and CMP in terms of power and performance. Donald et al [16] studied the temperature issues for SMT and CMP.

Burns et al [6] studied the area and clock effects of these two architectures in detail. The area of each component is calculated by understanding how they scale with increasing issue width and increasing number of supported threads. It is found that SMT gives best single thread performance but CMP achieves higher throughput.

It is clear that, in the SMT-CMP comparison, their relative behavior is different for different kinds of

workloads. In this paper, we perform a similar equal-area comparison between SMT and CMP processors to understand their behavior under TLS workloads.

2 Support Speculative Threads on SMT and CMP

There have been many proposals that extend existing cache coherence protocols to support TLS on CMP. In this paper, Our CMP-based TLS (CMP-TLS) design builds on top of the STAMPede approach[35], except for the replacement of the crossbar interconnect—a bus interconnect is used to connect the first level caches with the unified second level cache. Our SMT-based TLS (SMT-TLS) [29], augments a SMT architecture with extra bits in the unified first level cache to track data dependence violations and buffer speculative states In this section, we will briefly discuss how the execution patterns of TLS are different on these two architectures:

Buffering Speculative values In the CMP-TLS, when a speculative thread executes a store instruction, the speculative value is buffered in the private L1 cache and a special SM (speculatively modified) bit is set. This value is maintained in the cache and never evicted until the thread commits. At thread commit, the value becomes part of the non-speculative state and the SM bit is cleared. So to support speculative buffering, CMP-TLS needs just one SM bit per cache line (or one bit per word to avoid multiple-writer problem described in [34]). The L2 cache is always maintained as non-speculative.

Also in SMT-TLS, when the speculative threads execute a store instruction, the value has to be buffered in L1 cache. But since threads share the same cache and there may be more than one active speculative thread, the cache must be able to support multiple versions of the same value each from a different speculative thread. The SMT-TLS uses the associativity of the cache to support the different versions. The cache line in the SMT data cache uses additional bits (one bit per thread as in [29]) in addition to the SM bit to record the version number. So the SMT-TLS has a slightly more overhead in cache.

Both the CMP-TLS and SMT-TLS use a special SL bit per cache line to track inter-thread dependencies.

Overflowing Speculation Buffer As we saw above, the hardware maintains speculative values and dependence information by using special bits in the cache. So to ensure correctness, the cache lines with the special (SL and SM) bits set should not be evicted. But capacity and conflict misses occur in the cache and

we may be forced to evict the line with SL or SM bit set. In CMP-TLS where the cache is private, when we are faced with this situation for a speculative thread, we just force the speculative thread causing the eviction to wait till it becomes non-speculative. The cache of the non-speculative thread does not have any of the SL or SM bit set, so we do not have this issue for the non-speculative thread.

The SMT-TLS uses the associativity of the cache to hold the different versions. So if there are two speculative values mapped to the same set in the cache, the cache may not have enough associativity to hold all the versions of these two values. Here also we can make the speculative thread causing the cache miss to wait till it becomes non-speculative. But if the non-speculative thread is causing the eviction, we cannot make it wait (can cause deadlock). So we are forced to evict some cache line with SL or SM bit set. To ensure correctness the corresponding speculative thread(s) need to be squashed. This type of squash due to overflow occurs only for SMT-TLS. So it is possible that the SMT-TLS have more squashes than the CMP-TLS

Synchronizing Parallel Threads Some inter-thread data dependences are known during compile time so it is profitable to synchronize for such dependencies. In CMP-TLS when the thread encounters a wait operation, it just stalls without executing any more instructions. This is the same for SMT-TLS. However, since multiple critical resources in SMT are shared, a stalled thread holding on resources is likely to degrade the performance of other threads. Hence, in SMT-TLS, when a speculative thread is waiting on synchronization, the shared resources occupied by it are reclaimed.

3 Equal chip-area CMP and SMT processors

For both the CMP-TLS and the SMT-TLS design, many configurations are possible. To fairly compare these two designs, we must identify a common cost basis. In this paper, the cost basis is chip area. We give both architectures equal chip area and compare their performance and power efficiency under this constraint.

3.1 Deciding of configurations

Table 1: Processor parameters

Parameter	Superscalar and SMT	CMP
Fetch Width	6 instructions	3 instructions
Decode, issue and commit width	6 , 4 and 4 instructions	3,2,2 instructions
Function Units		
Integer adders	Superscalar: 4 adders, SMT: 3 adders	2 adders
Multipliers	Superscalar: 2, SMT: 1	1 multipliers
Memory ports	2 read and 1 write ports	1 read and 1 write ports
Latency	1 cycle for integer, 12 cycle for floating point	
Register Update Unit	128 entries	64 entries
(ROB,issue queue)		
LSQ size	64 entries	32 entries
L1D Cache	64K, 4 way associative, 32B blocksize	16K, 4 way, 32B blocksize
L1I Cache	64K, 4 way associative, 32B blocksize	16K, 4 way, 32B blocksize
Bus width		128 bits, 1 cycle latency
	Common to Superscalar, SMT and CMP	
Unified L2	2MB, 8 way associative, 64B blocksize	
Cache Latency	L1 1 cycle, L2 18 cycles	
Memory latency	150 cycles for 1st chunk, 18 cycles subsequent chunks	
Branch predictor	Bimod, 2K entries	
Branch mis-prediction penalty	6 cycles	
Physical registers per thread	128 Integer, 128 Floating point and 64 predicate registers	
Thread overhead	5 cycles fork, 5 cycles commit and 1 cycle inter-thread communication	

Like is most other studies [34, 31, 37, 7, 43] we assumed a four-core CMP for CMP-TLS. To make the area of CMP-TLS equal to that of Superscalar, we need to reduce the size of each core in CMP to almost one fourth of that of Superscalar. For some processor structures like the reorder buffer and LSQ, the area increases 4 times for every 2 times increase in the number of entries. Other hardware structures like function units, scale linearly with increase in number of function units. By taking these factors into consideration, the size of the CMP core is reduced so that the total CMP-TLS area matches the area of the Superscalar configuration. For SMT-TLS, the increase in area is mainly due to the increased complexity due to addition of threads (4 additional threads). To offset this increase in area, the number of function units in SMT-TLS is reduced. We tune the different parameters till the three configurations became equal in area. The final architectural parameters of the Superscalar, SMT-TLS and CMP-TLS designs are shown in Table 1. The following sections explain how we estimated the area of the different configurations.

To estimate area, we use the tool presented in [26]. The tool takes as input the different processor parameters like the cache size, number of function units, issue width and it outputs the area estimate in terms of λ^2, where λ is defined as half the minimum feature size. For array structures, such as caches, registers, reorder buffer and branch predictor, it first estimates the number of bits and then number of ports.

Table 2: Summary of area estimation

Hardware structures	Area effect due to			Area in $M\lambda^2$			
	Issue width (d)	SMT Threads(t)	Function units(f)	Superscalar	SMT	CMP/core	CMP full
Function units							
Integer units	None	None	O(f)	607.91	379.94	303.95	1215.82
Floating point units	None	None	O(f)	400.00	200.00	100.00	400.00
Load store units	None	None	O(f)	450.00	450.00	300.00	1200.00
				1457.91	1029.94	703.95	2815.82
Pipeline logic:							
Fetch unit	O(d)	O(t)	None	195.00	780.00	97.50	390.00
Decode unit (dispatch)	O(d)	20% overhead	None	180.00	720.00	90.00	360.00
Issue (scheduler)	O(d)	None	None	160.00	160.00	80.00	320.00
Write back unit	None	None	O(f)	260.00	180.00	120.00	480.00
Commit unit	O(d)	20% overhead	None	88.00	96.80	44.00	176.00
				883.00	1441.60	431.50	1726.00
Register File	$O(\min(f,d))^2$	O(t)	$O(\min(f,d))^2$	381.29	1183.91	113.39	453.57
LSQ	None	None	$O(f^2)$	451.54	305.66	70.96	283.85
RUU	None	None	$O(f^2)$	3158.95	1985.59	368.05	1472.20
BTB, ALAT, IFQ				109.85	121.00	29.43	117.72
Caches							
TLBS	No change	Extra port	No change	208.76	229.64	84.58	338.32
Level 1 i-cache	No change	Extra port	No change	1426.72	1956.64	358.97	1435.89
Level 1 d-cache	No change	Extra TLS bits	O(ports)	1956.64	2056.76	377.22	1508.89
Level 2 cache	No change	No change	No change	41397.78	41397.78		41397.78
Total Area				51432.45	51708.43		51550.04
Total chip area in mm^2 $for 70nm technology$				63.00	63.34		63.15

The area is then calculated by multiplying the cell height and cell width with the number of bits. For other structures such as the fetch unit and the function units, it gives an empirical estimate of the area. We modified this tool to estimate the area of SMT and CMP models. A summary of area estimation is given in Table 2. For fair comparison the size of the L2 cache is maintained the same across the three configurations.

3.2 Estimating the Area of A SMT Processor

Function units. The SMT uses slightly fewer number of function units than the Superscalar in order to compensate for its increased area due to additional complexity. Hence the function units in SMT-TLS consume about 40% less area.

Pipeline logic: The fetch unit of SMT has to maintain a separate program counter and call stack for each thread and also it implements the ICOUNT fetch policy [38]. Due to these additional complexities, we assume that the area of the fetch unit is the number of threads times the superscalar fetch unit area

(as in [5]).

In a SMT processor, the dispatch (decode) of instructions into the reorder buffer has to be done in order for each thread. Also, the commit stage has to commit the instructions from each thread in order. And if any thread cannot commit in a particular cycle, the commit logic should commit threads from the other threads. Due to this additional complexity, we assume that the dispatch and decode stage consume about 20% more area than the corresponding superscalar stages. The other stages do not distinguish between the threads, so there is no additional complexity when compared to the superscalar processor.

The complexity of writeback stage depends on the number of write backs possible within each cycle and this depends on the number of function units. So the writeback stage of superscalar consumes about 40% more area than SMT due to the extra function units.

On the whole, there is an overhead of about 63% for SMT processor due to pipeline logic.

Register file: The area of the register file depends on the number of registers and on the number of read and write ports. The number of read ports to register file is usually equal to (issue width of the processor * number of operands per instruction). But if the number of function units is smaller than the issue width, we can reduce the number of read ports (we will not need the extra ports). The number of write ports is equal to the commit width.

Since the SMT uses fewer function units, the register file of SMT has a smaller number of ports. So the SMT register file uses about 28% less area than that of Superscalar. But SMT has four threads and a separate register file is needed for each thread. So, in effect the SMT uses about 3 times chip area for its register file than the Superscalar.

RUU and LSQ. The area of register update unit (RUU, it combines reorder buffer, rename registers and issue queue) and the load/store queue (LSQ) depend on their number of entries as well as the number of ports from the function units. Due to the reduced number of function units, RUU/LSQ of SMT consumes about 60% less area.

Caches. In SMT, every cycle a thread is selected according to the fetch policy. If the selected thread is not able to use all of the fetch width, the next thread can fetch its instructions as in [38]. To support this,

we need an extra port. This leads to about 40% increase in area for level 1 cache of SMT.

Also the level 1 data cache has some extra bits to support TLS as shown in section 3. In total, we add 16 bits per cache line. The area increase due to these bits is only about 5%.

3.3 Estimating the Area of A CMP Processor

Function units: The smaller CMP core uses fewer number of function units and thus uses 50% lesser area than the Superscalar configuration. But for the four cores, the area would be 2 times that of Superscalar. The area used could be further reduced by reducing the number of function units but that could have a drastic effect on performance, so we have to compensate for this increase in area in other parts of the configuration.

Pipeline logic: The fetch, decode, issue and commit width of the CMP core is half of that of the Superscalar. Due to this we gain about 50% reduction in area for pipeline logic, but for four cores we need 2 times the area.

Register file: The register file of CMP has the same number of bits as that of superscalar. But the number of read and write ports to the register file is much lesser due to it fewer function units. Hence, So each register file is about 3 times smaller in area than the one used for Superscalar. But this gain has been offset as we use four cores.

RUU and LSQ: Every function unit has read and write ports to the RUU and LSQ, and each such port increases both the height and width of every cell in the RUU and LSQ. So, any increase in the number of function units leads to a square increase in the area of RUU and LSQ. Also, each CMP core uses only half the number of RUU and LSQ entries. Due to this, the area consumed by each core of CMP is about 8 times smaller than that of Superscalar. With this gain in area we are able to compensate the increase in area due to other effects.

Caches: The size of each cache of CMP is one fourth of that of Superscalar and SMT. Each level 1 data cache of CMP has lesser number of read ports. Due to this the area occupied by caches (both i-cache and d-cache) for CMP is about 15% smaller than that of Superscalar. Like SMT based a approach, the CMP approach also uses extra bits in cache to support TLS.

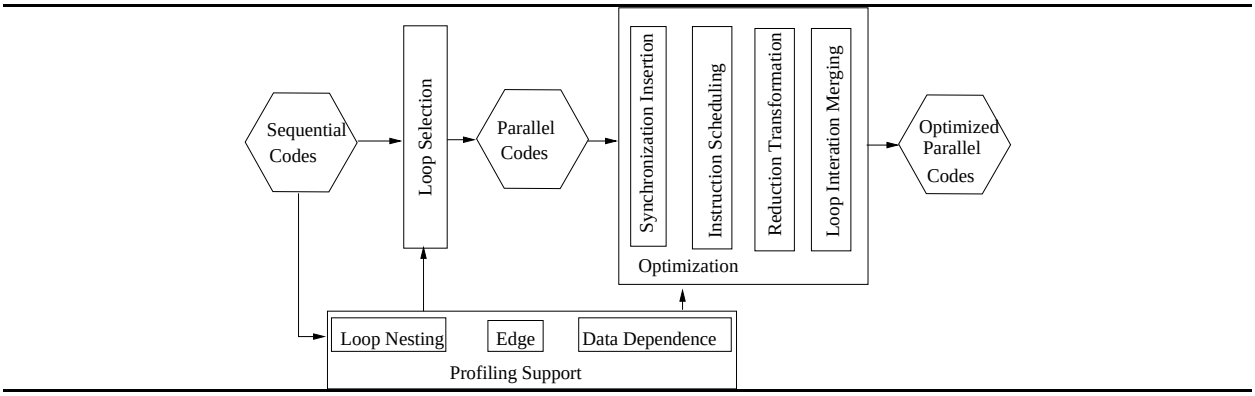

Figure 1: Compilation infrastructure

4 Evaluation Methodology

To quantitatively evaluated the power efficiency of supporting speculative threads on both the SMT and the CMP architecture, we parallelize a set of benchmarks from the SPEC2000 benchmark suite and simulate the execution of these benchmarks on a detailed architectural simulator augmented with a power model. In the rest of this section, we will describe the compilation and simulation infrastructure used in this paper.

4.1 Compilation Infrastructure

Our compiler infrastructure is built on Intel's Open Resource Compiler(ORC) [10], an industrial-strength open-source compiler targeting Intel's Itanium Processor Family (IPF).

To create efficient speculative parallel threads, compiler must perform accurate performance trade-off analysis to determine whether the benefit of speculative parallel execution outweighs the cost of failed speculation. In our case, the compiler performs such analysis based on loop nesting, edge, as well as data dependence profiling, as shown in Figure 1.

The parallel compiler has two distinct phases, as shown in Figure 1, loop selection and optimization:

Loop Selection: In the loop selection phase, the compiler estimates the parallel performance of each loop based on the cost of synchronization, as well as the probability and cost of speculation failure. The compiler then choose to parallelize a set of loops that maximize the overall program performance based on such estimation [42].

Optimization: The selected parallel loops are optimized with various compiler optimization techniques to enhance parallel performance. In our case, the following optimizations are applied: (i) all register-

Table 3: Details of Benchmarks

Benchmark	No of loops selected	coverage of selected regions
equake	1	95%
parser	40	89%
art	14	99%
bzip2	2	88%
mcf	7	98%
mesa	2	80%
vpr	2	55%
twolf	8	54%

resident values and memory-resident values that cause inter-thread data dependences in 20% of all threads are synchronized [44]; (ii) instructions are scheduled to reduce the critical forwarding path introduced by the synchronization [43, 41]; (iii) computation and usage of reduction-like variables are transformed to avoid speculation failure and reduce synchronization [41]; and (iv) consecutive loop iterations are merged to balance the workload of neighboring threads.

4.2 Simulation Infrastructure

We evaluate the different configurations using a detailed out-of-order superscalar simulator based on Simplescalar 3.0. We build the SMT and CMP based TLS support on top of this simulator. The configurations we simulate are detailed in Table 1. Our simulator is a trace-driven simulator and the traces for the benchmarks are generated using the Pin instrumentation tool[25].

We integrated our simulator with the power model used in Wattch [4] to simulate power consumption. We assume 70nm technology and scale the power parameters accordingly. All the structures added to implement TLS such as the signal table, owner buffer and the squash logic are modeled and their power consumption taken into account. We simulate the common bus between the multiple cores, and the power for the bus is modeled using orion[40].

4.3 Benchmarks

We use the benchmarks from SPEC2000 integer suite for evaluation. All simulations are performed using the ref input set. As in many other studies [34], for each benchmark, a total of 1 billion instructions are simulated. Also we fast-forward the simulator for several million instructions to skip the initialization part

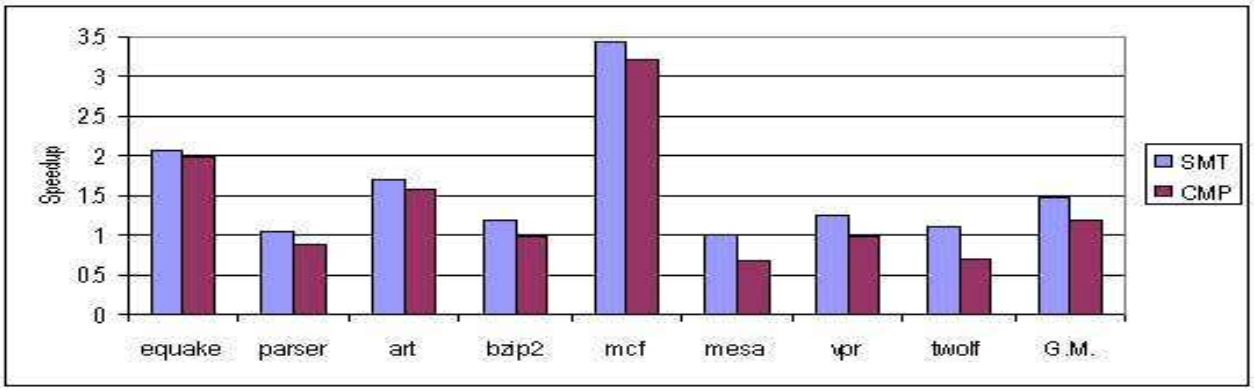

Figure 2: Speedup of entire simulated region

and to reach the parallelized regions. Table 3 describes the details of the benchmarks simulated. (Note: we are running more benchmarks, and will have more complete results for all Spec2000 benchmarks in the final version of the paper) The coverage indicates the coverage of the selected loops within the simulated region on 1 billion instructions.

5 Performance and Power Comparisons

We compare the three different configurations - Superscalar, SMT and CMP in terms of performance in Section 5.1 and power consumption in Section 5.2. We also use the energy-delay product (EDP) metric to combine the effect of both power and speedup in Section 5.3. Finally, in Section 5.4, we study the impact of increasing the clock rate for CMP.

5.1 Performance Comparison

Fig. 2 shows the speedup of the entire simulated (1 billion instructions) region which includes both parallel and sequential portions. We can see that for all benchmarks except mesa, the SMT outperforms the Superscalar architecture. The geometric mean shows that it can achieve about 46% speedup over Superscalar. The CMP showed speedup only for equake, art and mcf. It achieved about 19% average speedup. To better understand the performance variation, let us look at Fig. 3 which gives the breakdown of cycles while executing only the parallelized regions.

For benchmark mcf, we can see from Fig. 3 that the Superscalar wastes most of the time waiting for memory due to high cache miss rate. In SMT and CMP, the speculative threads run in parallel and issue

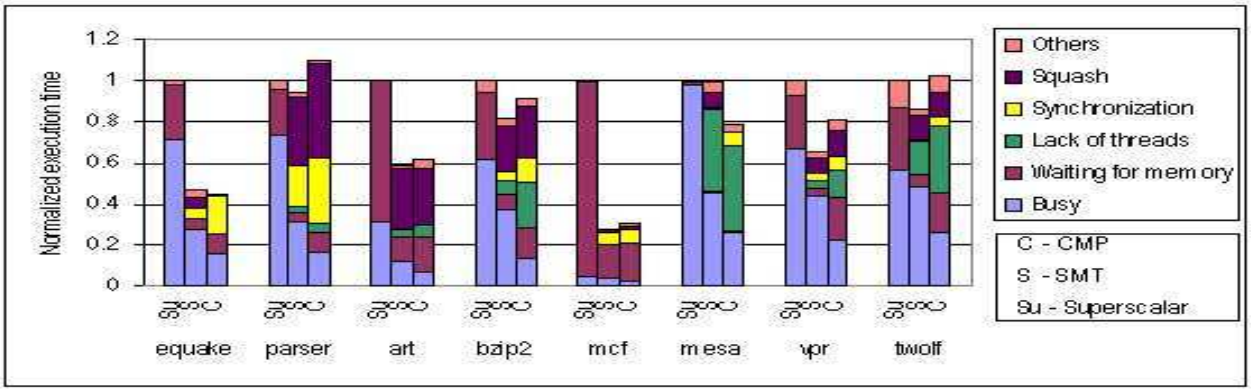

Figure 3: Breakdown of execution cycles for parallel region. All bars normalized to Superscalar

memory operations earlier and the cache misses from different threads overlap each other. Due to this memory level parallelism, the memory waiting time is much less in SMT and CMP. Time wasted due to thread mis-speculation is very little for mcf, so we gain from thread level parallelism. Due to the combined effect of memory level parallelism and thread level parallelism, mcf gains almost 3 times performance for both SMT and CMP architectures. Similar results for mcf have been observed in [31].

The benchmark equake also has very good thread level parallelism leading to almost 2 times performance for SMT and CMP. Even though the equake loops were completely parallel, the SMT suffered from some false squashes. This is due to the buffer overflow effect we discussed in section 3.3. When a thread requests a cache line occupied by a later thread (more speculative), the later thread is squashed. This effect is not seen on CMP due to its private cache.

In benchmark art, the Superscalar suffers a large number of cache misses leading to a significant memory waiting time. The SMT and CMP were able to overcome this due to their parallelism. But one of the loops selected in art, the SMT and CMP both had many squashes. In this loop, adjacent elements in a loop are accessed and since we maintained only one SL bit per cache line as shown in section 3.2, we suffered from many false sharing violations. Since the memory waiting time is the main performance bottleneck for art, SMT and CMP still manage to achieve very good performance.

In benchmark twolf, the CMP shows a slowdown of about 3%. From 3 we can see that a significant amount of time is wasted due to lack of threads. In twolf, many selected loops have very small iteration count, due to this some cores in the CMP are not active. But in SMT, even though the number of available threads is smaller, the available threads are able to use all the processor resources efficiently. A similar effect

can be found for benchmarks vpr, bzip and mesa.

In mesa, both SMT and CMP suffer significantly due to lack of threads. Also, the SMT has only half the number of floating point units than Superscalar. The SMT also suffered from false squashes due to buffer overflow. Due to these effects, SMT suffered about 1% performance loss. In benchmark parser, both SMT and CMP suffered from many thread squashes. This lead to about 13% slowdown for CMP and SMT gained only 5% speedup. For benchmark bzip2, the SMT achieves about 20% speedup, but CMP gets only 2% speedup. This is because, some loops selected have low iteration count and some of the cores in CMP remain idle.

In benchmark vpr, CMP had about 24% speedup over Superscalar for the parallel region as shown in 3. But 2 shows about 2% loss in performance. This is because the coverage of parallel region for vpr is only about 55%. While executing non-parallelized regions, CMP can use only one core and since this core is much smaller than that of Superscalar and SMT, it suffers significant performance loss. This effect can also be seen for benchmarks twolf which has only 54% coverage. In benchmark mesa, CMP gained about 20% speedup from the parallel regions as shown in 3. But it slowed down by significantly (about 70%) while executing non-parallelized regions. This is because the CMP core has only one floating point unit and this became a bottleneck while executing the non-parallelized code (which has about 20% coverage).

Summary. For benchmarks with significant non-parallelized regions CMP shows a significant performance loss. Also while executing loops with low iteration count, the CMP does not use all its cores leading to performance loss for these loops. The SMT does not suffer from any of these but shows increased squashes due to buffer overflow. The impact of extra squashes is less on speedup and the SMT showed good performance for almost all the benchmarks.

5.2 Power Comparison

The Wattch simulator produces different results for power consumption depending on the assumption made for clock gating. We can assume a perfect clock gating, i.e., a transistor consumes zero power when idle. So the only power consumed is the power due to activity in the processor. This gives dynamic power consumption but it is not a realistic estimate of actual power consumption, because a transistor consumes some static power while being idle. If we assume no clock gating, it means the processor always uses the same amount of power independent of the activity of the processor. This is also not an accurate measure

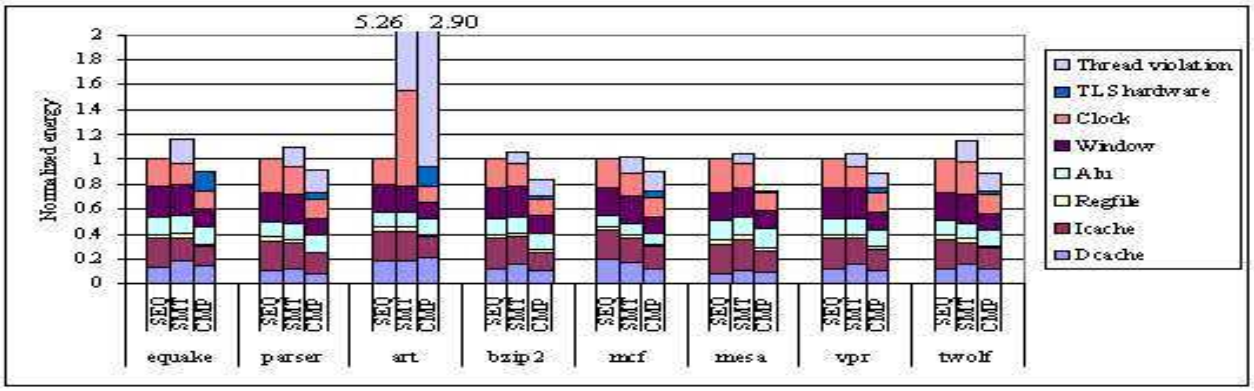

Figure 4: Breakdown of dynamic power consumption normalized to Superscalar

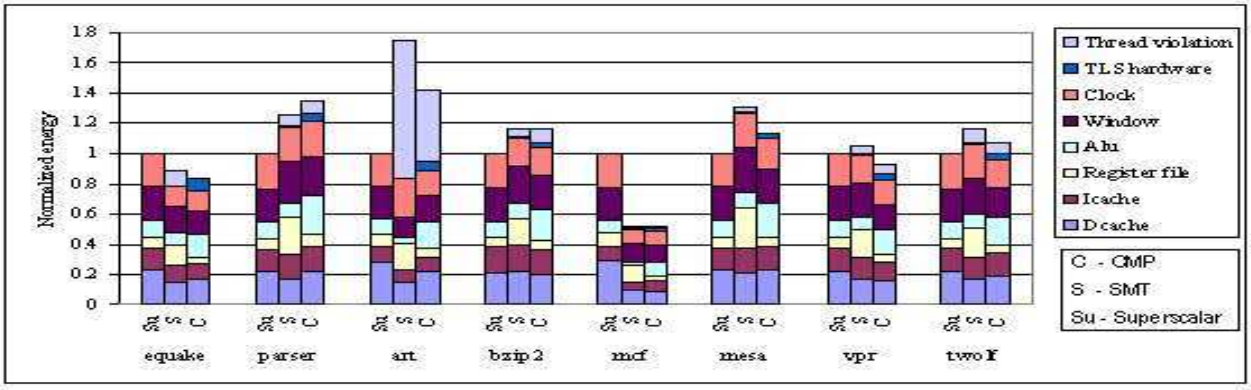

Figure 5: Breakdown of total power consumption which includes both dynamic and static power. All bars normalized to Superscalar

for total power consumption. In our simulation we assume an aggressive clock gating scheme where the hardware structures consume 10% of the dynamic power when they are not used.

To better understand power behavior of the different architectures, we will first analyze the dynamic power (cc2 in Wattch) separately and later show the total power consumption including static power(cc3 in Wattch).

5.2.1 Dynamic power

Fig. 6 shows the breakdown of dynamic power consumption of the three different configurations for the entire execution of the simulated regions. The figure shows a scaled power consumption where the power consumption of SMT and CMP are scaled to the Superscalar configuration.

The dynamic power is proportional to αC.V^{2}f, where α is the activity factor, C is the capacitance of the

transistor, V the supply voltage and f the frequency of the circuit. In our simulation, we kept the V and f term same for all three configurations. So dynamic power differences among the three configurations are mainly due to the activity factor or the capacitance of the circuit. The Superscalar and SMT have similar configurations, so their C value is almost the same for most hardware structures. But the CMP configuration uses smaller structures and hence has a smaller C value than Superscalar and SMT. The work done by all three configuration is very similar (all execute 1 billion instructions), so the activity factor should remain the same. For example, the number of additions and hence the number of times the adder circuit is involved for the 1 billion instructions is the same for all three configurations. But the TLS architectures execute threads speculatively and some of them need to be squashed. This extra work causes extra activities in the processor and consumes extra dynamic power.

In section section 3.1, we saw that the SMT and CMP maintain only one SL bit per cache line to keep track of inter-thread dependency. But with just one bit, we cannot tell which particular word was speculatively accessed in a cache line. Due to this coarse-grained dependency tracking, the TLS architectures could suffer from false dependency violations. We see this effect for benchmark art, when the false squashes occur inside an inner loop. So for each iteration of the outer loop, the squash happens multiple times. In effect, the actual work done is many times more than the required work. This leads to a huge amount of power wasted due to speculation for both SMT and CMP.

Also, in benchmarks equake and mesa, the number of squashes suffered by SMT is more due to false squashes caused by buffer overflow. Due to this effect SMT wastes more power than CMP for these benchmarks.

Both CMP and SMT architectures use extra logic to implement TLS. The power consumed by such extra hardware is shown in Fig. 6 as ”TLS hardware”. This includes the power for signal table, the common bus in the case of CMP and other special structures. This extra power is very small for SMT, but CMP consumes more power due to the common bus between cores.

The data cache power in Fig. 6 includes the level 2 cache power which forms about 20% of the data cache power. The low power consumption in the level 2 cache is because of its very low activity. The level 1 data cache and instruction cache consume a large portion of the total dynamic power due to their large sizes and high activities. Especially, the instruction cache consumes a significant amount of dynamic power since it is accessed almost every cycle. The SMT and Superscalar consume similar power for the caches but

the CMP consumes less power due to its smaller cache. The smaller level 1 caches can lead to increased accesses to level 2 cache, but this effect was relatively small.

The window power includes the hardware structures used inside the processor pipeline like reorder buffer, load store queue, etc. These structures usually have a very high activity factor α and also a large C value. The CMP, due to its smaller structure, consumes much less power than SMT and CMP.

The number of accesses to function units is similar in all the three configurations. Since the function units are accessed individually, the CMP does not gain any power due to its smaller number of per core function units.

The clock power is proportional to the dimensions of the die and the activity in the processor. In benchmark art, due to the increased activity from speculation, the power wasted in clock also increases.

5.2.2 Total power

The total power consumption includes static power(leakage power) in addition to the dynamic power we analyzed in the last section. The static power depends on the number of transistors in the circuit and the time of execution. If the number of transistors increases, we have more sources of leakage. When we use more time to run the program, the amount of leakage will also be more.

For almost all benchmarks, the SMT processor achieves better speedup than the Superscalar and CMP processor. So it takes less time to execute which makes its leakage power much less than the other configurations. Due to this reduced leakage, it is able to offset much of the dynamic power increase. For benchmarks like mcf and equake, where we get very good performance, the total power is less than that of Superscalar. In benchmark art, from Fig. 6 we see a five times increase in dynamic power for SMT. But from Fig. 5, we see that this increase is offset by the gain in static power and this results in only about 50% increase in total power. Similarly, the benchmark equake for which we had about 15% increase in dynamic power, we get about 14% gain in total power due to the increase in speedup. Similar effect is seen for CMP but the decrease in static power is limited due to its reduced speedup and amount of static power overhead.

The register file of SMT is four times larger than that of Superscalar and similar in complexity, so the total power consumed is more in the register file of SMT. The increase is mainly due to the increase in leakage power. The CMP also has replicated register file, but the register file is less complex with fewer

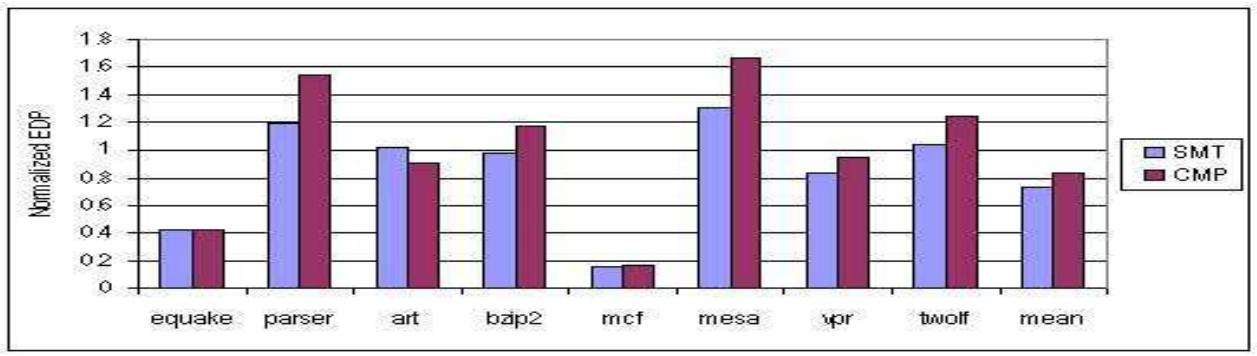

Figure 6: EDP comparison showing the relative efficiency of TLS architectures. All bars normalized to Superscalar

number of ports. Due to this, the CMP shows only a small increase in register file leakage.

The CMP uses fewer number of function units per core. But if we consider all cores, the number of function units is much larger than other configurations. This leads to the increase in leakage power for function units in CMP.

Thread level speculation wastes power but this increases only the dynamic power. So the effect of this wasted power on the total power consumption is minimal.

The CMP processor, while executing sequential region of code, does not use more than one core. We assume that the unused cores can be switched off completely so that they do not consume any leakage power. This switching off and on of the cores involves power and performance overhead, which we did not consider. So the actual power consumption of CMP will be slightly higher than the power shown in Fig. 5.

Summary. The dynamic power consumption of SMT is more than that of CMP due to its complex cores. But the static power is less than the other configurations due to SMT's increased speedup. With this gain, the SMT was able to recover some of its losses in dynamic power. Both SMT and CMP suffer from extra dynamic power due to speculative thread execution. Inspite of this power wastage, the CMP due to its smaller cores does not have any power overhead over Superscalar.

5.3 *EDP* Comparison

We know from the previous sections that the three configurations have different behavior in speedup and power consumption. To compare them using one metric, we need to combine the effect of speedup and power consumption. Fig. 6 uses the energy-delay product (EDP) metric which takes the product of power

consumption and execution time.

From the figure we can see that, for all benchmarks the SMT has the lowest energy-delay product. It shows about 27% better results than the Superscalar processor, and the CMP is only about 17% better than the Superscalar processor.

5.4 Impact of Increasing Clock Frequency

In the previous sections, we assumed that all three configurations use the same clock frequency. The Superscalar and SMT have similar complexity, so their clock frequency should be the same. The CMP core is simpler, hence we could run it at a higher frequency. But increasing frequency will cost power and the energy-delay product (EDP) would approximately remain the same. So the SMT still be better than CMP in terms of EDP.

5.5 Thread overhead

In our experiments we assumed the same overhead for thread forking, committing and inter-thread communication, for both SMT and CMP configurations. In CMP, since the threads are located in different cores the overhead could be higher than the SMT. But in our experiments we found that these thread overheads has minimal impact on performance. The fork overhead and inter-thread communication overhead were small and we found that they are easily hidden due to out-of-order execution. The commit overhead mainly depends on the amount of data to be committed which is the same for both SMT and CMP.

6 Conclusions

From our experiments, we are able to draw the following conclusions on the performance of SMT and CMP under the context of TLS: the larger and more complex structure of SMT allows to explore TLP and ILP simultaneously, while CMP with smaller cores can only efficiently exploit thread-level parallelism. This gives SMT a performance edge over CMP when parallel threads cannot be extracted. Thus, SMT outperforms CMP by approximately 23% for SPEC2000 benchmarks. On the other hand, the smaller cores and caches in CMP can potentially make it more power efficient. We have observed that the CMP processor consumes 33% less dynamic power than SMT. However, as on-chip device size decreases, static power

is becoming more important. When static power consumption is taking into consideration, the power-advantage of CMP diminishes. Our results show that when leakage power is taken into consideration, SMT consumes only 8% more power than CMP.

References

[1] AKKARY, H., AND DRISCOLL, M. A Dynamic Multithreading Processor. In *31st Annual IEEE/ACM International Symposium on Microarchitecture (Micro-31)* (December 1998).

[2] AMD CORPORATION. Leading the Industry: Multi-core Technology & Dual-Core Processors from AMD. http://multicore.amd.com/en/Technology/, 2005.

[3] BHOWMIK, A., AND FRANKLIN, M. A Fast Approximate Interprocedural Analysis for Speculative Multithreading Compiler. In *17th Annual ACM International Conference on Supercomputing* (2003).

[4] BROOKS, D., TIWARI, V., AND MARTONOSI, M. Wattch: a framework for architectural-level power analysis and optimizations. In *27th Annual International Symposium on Computer Architecture (ISCA '00)* (2000).

[5] BURNS, J., AND GAUDIOT, J.-L. Quantifying the smt layout overhead-does smt pull its weight? In *6th International Symposium on High-Performance Computer Architecture (HPCA-6)* (2000), pp. 109–120.

[6] BURNS, J., AND GAUDIOT, J.-L. Area and system clock effects on smt/cmp throughput. *IEEE Trans. Computers 54*, 2 (2005), 141–152.

[7] CINTRA, M., AND TORRELLAS, J. Learning Cross-Thread Violations in Speculative Parallelization for Multiprocessors. In *8th International Symposium on High-Performance Computer Architecture (HPCA-8)* (2002).

[8] DU, Z.-H., LIM, C.-C., LI, X.-F., YANG, C., ZHAO, Q., AND NGAI, T.-F. A Cost-Driven Compilation Framework for Speculative Parallelization of Sequential Programs. In *ACM SIGPLAN 04 Conference on Programming Language Design and Implementation (PLDI'04)* (June 2004).

[9] DUBEY, P., O'BRIEN, K., O'BRIEN, K., AND BARTON, C. Single-Program Speculative Multithreading (SPSM) Architecture: Compiler-assisted Fine-Grained Multithreading. In *International Conference on Parallel Architectures and Compilation Techniques (PACT 1995)* (June 1995).

[10] FOR ITANIUM PROCESSORS, O. R. C., Jan 2003.

[11] FRANKLIN, M., AND SOHI, G. S. The expandable split window paradigm for exploiting fine-grain parallelsim. In *19th Annual International Symposium on Computer Architecture (ISCA '92)* (May 1992), pp. 58–67.

[12] GUPTA, M., AND NIM, R. Techniques for Speculative Run-Time Parallelization of Loops. In *Supercomputing '98* (November 1998).

[13] HAMMOND, L., WILLEY, M., AND OLUKOTUN, K. Data Speculation Support for a Chip Multiprocessor. In *Proceedings of ASPLOS-VIII* (October 1998).

[14] INTEL CORPORATION. Intel's Dual-Core Processor for Desktop PCs. http://www.intel.com/personal/desktopcomputer/dual_core/, 2005.

[15] INTEL CORPORATION. Intel Itanium Architecture Software Developer's Manual, Revision 2.2. http://www.intel.com/design/itanium/manuals/iiasdmanual.htm, 2006.

[16] J.DONALD, AND M.MARTONOSI. Temperature-aware design issues for smt and cmp architectures. In *Fifth Workshop on Complexity-Effective Design (WCED) in conjunction with ISCA-31* (June 2004).

[17] JOHNSON, T., EIGENMANN, R., AND VIJAYKUMAR, T. Min-Cut Program Decomposition for Thread-Level Speculation. In *ACM SIGPLAN 04 Conference on Programming Language Design and Implementation (PLDI'04)* (June 2004).

[18] KALLA, R., SINHAROY, B., AND TENDLER, J. M. IBM Power5 Chip: A Dual-Core Multithreaded Processor. *Microprocessor Forum '99* (October 1999).

[19] KAXIRAS, S., NARLIKAR, G. J., BERENBAUM, A. D., AND HU, Z. Comparing power consumption of an smt and a cmp dsp for mobile phone workloads. In *CASES* (2001).

[20] KNIGHT, T. An Architecture for Mostly Functional Languages. In *Proceedings of the ACM Lisp and Functional Programming Conference* (August 1986), pp. 500–519.

[21] KRISHNAN, V., AND TORRELLAS, J. The Need for Fast Communication in Hardware-Based Speculative Chip Multiprocessors. In *International Conference on Parallel Architectures and Compilation Techniques (PACT 1999)* (October 1999).

[22] LI, Y., BROOKS, D., HU, Z., AND SKADRON, K. Performance, energy, and thermal considerations for smt and cmp architectures. In *11th International Symposium on High-Performance Computer Architecture (HPCA-11)* (2005).

[23] LIU, W., TUCK, J., CEZE, L., AHN, W., STRAUSS, K., RENAU, J., AND TORRELLAS, J. POSH: A TLS Compiler that Exploits Program Structure. In *ACM SIGPLAN 2006 Symposium on Principles and Practice of Parallel Programming* (March 2006).

[24] LO, J., EGGERS, S., EMER, J., LEVY, H., STAMM, R., AND TULLSEN, D. Converting Thread-Level Parallelism Into Instruction-Level Parallelism via Simultaneous Multithreading. *ACM Comput. Surv.* (Aug. 1997), 322–354.

[25] LUK, C.-K., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., WALLACE, S., REDDI, V., AND HAZELWOOD, K. Pin: building customized program analysis tools with dynamic instrumentation. In *ACM SIGPLAN 05 Conference on Programming Language Design and Implementation (PLDI'05)* (June 2005).

[26] MARC, S., REINER, K., L., L. J., THEO, U., AND MATEO, V. Transistor count and chip-space estimation of simplescalar-based microprocessor models. In *Workshop on Complexity-Effective Design, in conjunction with the 28th International Symposium on Computer Architecture* (June 2001).

[27] MARCUELLO, P., AND GONZALEZ, A. Clustered Speculative Multithreaded Processors. In *13th Annual ACM International Conference on Supercomputing* (Rhodes, Greece, June 1999).

[28] OPLINGER, J., HEINE, D., AND LAM, M. In Search of Speculative Thread-Level Parallelism. In *Proceedings PACT 99* (October 1999).

[29] PACKIRISAMY, V., WANG, S., A.ZHAI, HSU, W.-C., AND YEW, P.-C. Supporting speculative multithreading on simultaneous multithreaded processors. In *12th International Conference on High Performance Computing HiPC'2006* (Bengaluru. India, Dec. 2006).

[30] PARK, I., FALSAFI, B., AND VIJAYKUMAR, T. Implicitly-multithreaded processors. In *30th Annual International Symposium on Computer Architecture (ISCA '03)* (June 2003).

[31] RENAU, J., STRAUSS, K., CEZE, L., LIU, W., SARANGI, S. R., TUCK, J., AND TORRELLAS, J. Energy-efficient thread-level speculation. *IEEE Micro 26*, 1 (2006), 80–91.

[32] SASANKA, R., ADVE, S. V., CHEN, Y.-K., AND DEBES, E. The energy efficiency of cmp vs. smt for multimedia workloads. In *18th Annual ACM International Conference on Supercomputing* (2004), pp. 196–206.

[33] SOHI, G. S., BREACH, S., AND VIJAYKUMAR, T. N. Multiscalar Processors. In *22nd Annual International Symposium on Computer Architecture (ISCA '95)* (June 1995), pp. 414–425.

[34] STEFFAN, J. G., COLOHAN, C. B., ZHAI, A., AND MOWRY, T. C. A Scalable Approach to Thread-Level Speculation. In *27th Annual International Symposium on Computer Architecture (ISCA '00)* (June 2000).

[35] STEFFAN, J. G., COLOHAN, C. B., ZHAI, A., AND MOWRY, T. C. The stampede approach to thread-level speculation. In *ACM Trans. on Computer System* (August 2005), vol. 23, pp. 253–300.

[36] SUN CORPORATION. Throughput Computing—Niagara. http://www.sun.com/processors/throughput/, 2005.

[37] TSAI, J.-Y., HUANG, J., AMLO, C., LILJA, D., AND YEW, P.-C. The Superthreaded Processor Architecture. *IEEE Transactions on Computers, Special Issue on Multithreaded Architectures 48*, 9 (September 1999).

[38] TULLSEN, D., EGGERS, S., EMER, J., LEVY, H., LO, J., AND STAMM, R. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In *23rd Annual International Symposium on Computer Architecture (ISCA '96)* (May 1996).

[39] VIJAYKUMAR, T. N., AND SOHI, G. S. Task Selection for a Multiscalar Processor. In *31st Annual IEEE/ACM International Symposium on Microarchitecture (Micro-31)* (Nov. 1998).

[40] WANG, H. Orion: A power-performance simulator for interconnection networks, 2002.

[41] WANG, S. *Compiler Techniques for Thread-Level Speculation*. PhD thesis, University of Minnesota, 2007.

[42] WANG, S., YELLAJYOSULA, K. S., ZHAI, A., AND YEW, P.-C. Loop Selection for Thread-Level Speculation. In *The 18th International Workshop on Languages and Compilers for Parallel Computing* (Oct 2005).

[43] ZHAI, A., COLOHAN, C. B., STEFFAN, J. G., AND MOWRY, T. C. Compiler Optimization of Scalar Value Communication Between Speculative Threads. In *10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)* (Oct 2002).

[44] ZHAI, A., COLOHAN, C. B., STEFFAN, J. G., AND MOWRY, T. C. Compiler Optimization of Memory-Resident Value Communication Between Speculative Threads. In *The 2004 International Symposium on Code Generation and Optimization* (Mar 2004).