Understanding the Energy Efficiency of Simultaneous Multithreading

Yingmin Li†, David Brooks‡, Zhigang Hu††, Kevin Skadron †, Pradip Bose††
† Dept. of Computer Science, University of Virginia †† IBM T.J. Watson Research Center
‡ Dept. of Computer Science, Harvard University
{yingmin,skadron}@cs.virginia.edu, dbrooks@eecs.harvard.edu, {zhigangh,pbose}@us.ibm.com

Abstract

Simultaneous multithreading (SMT) has proven to be an effective method of increasing the performance of microprocessors by extracting additional instruction-level parallelism from multiple threads. In current microprocessor designs, power-efficiency is of critical importance, and we present modeling extensions to an architectural simulator to allow us to study the power-performance efficiency of SMT. After a thorough design space exploration we find that SMT can provide a performance speedup of nearly 20% for a wide range of applications with a power overhead of roughly 24%. Thus, SMT can provide a substantial benefit for energy-efficiency metrics such as ED^2. We also explore the underlying reasons for the power uplift, analyze the impact of leakage-sensitive process technologies, and discuss our model validation strategy.

Categories and Subject Descriptors

C.1 [**Computer Systems Organization**]: Processor Architectures

General Terms

Design

Keywords

Multithreading

1. INTRODUCTION

Simultaneous multithreading (SMT) [13] is a relatively new microarchitectural paradigm that has found industrial application [5, 7]. The promise of SMT is area-efficient throughput enhancement; however, the significant boost (10-40%) in instructions per cycle (IPC) is accompanied by an increase in power consumption. Since the area increase reported for SMT execution is relatively small (less than 5% per chip) [6], the main concern in next-generation SMT processor design is that of worst-case power and temperature characteristics.

ISLPED'04, August 9–11, 2004, Newport Beach, California, USA.

In this paper, we first describe SMT modeling extensions to Turandot/PowerTimer, a power-performance modeling toolkit developed around an extensively upgraded cycle -accurate, microarchitecture-level PowerPC simulator [1, 9]. PowerTimer allows us to understand the fundamental trade-offs between power and performance in single and multi-threaded modes of execution. We describe the methods used for extending the baseline, single-threaded energy models to an SMT architecture. One of the major requirements in developing a power-performance simulator is a robust validation methodology, and we briefly describe the validation methodology currently in place in the PowerTimer project.

There has been recent related work in understanding area efficiency and power issues in multithreaded processors. Burns and Gaudiot [4] consider the scalability of various resources in an SMT processor and perform a detailed study of the area overhead of SMT processors. Seng and Tullsen study several power-aware optimizations in the context of a fixed-resource multithreaded microprocessor [11]. In this work, we recognize that increased processor utilization in SMT machines will impact power, but not area, and we focus on understanding the fundamental power-performance efficiency of SMT rather than SMT-specific power optimizations.

This paper provides several major contributions. First, we present power modeling extensions to a single-threaded power performance simulator and describe our validation methodology. Next, we provide a thorough design space exploration to understand the performance benefits and power costs of SMT in the context of extensions to an existing POWER4-like microarchitecture. We conclude that SMT is a very power-efficient design paradigm in terms of ED^2 and can provide a 20% performance improvement for a varied mix of workloads with a power overhead of around 24%. There are several underlying reasons for power uplift (the same as power increase) in SMT machines, and we diagnose the uplift by analyzing the machine at the unit level. We also analyze the impact of future technologies where static leakage power is more significant, and we determine that the power overhead of SMT decreases with leakier process technologies because the power uplift due to utilization is marginalized by the larger fraction of leakage power. Finally, we discuss the sensitivity of our conclusions to our modeling assumptions.

2. PERFORMANCE AND POWER MODELING OF SMT

2.1 SMT Extension to the Performance Model

In this section, we describe the extensions that are added to our performance model for supporting SMT. A distinguishing feature of SMT is that execution units (FXU, FPU, etc) are usually shared among threads, thread-specific resources, such as program counters, are always duplicated, while the rest of resources (branch predictor, caches, etc) can either be shared or duplicated depending on design choices. Since all these resources are already modeled in the single-threaded base model, the extensions to SMT are straightforward. In addition to resource extensions, extra control logic is needed at various pipeline stages to decide which threads should go ahead, while others should be stalled on a given cycle. A simple policy that is commonly used is "round-robin", where the choice of the target thread is rotated sequentially (with wrap-around) among the available threads. This is the default thread selection policy implemented in the new SMT-enabled performance model. In future work, more sophisticated thread prioritization policies will be added and tested.

2.2 SMT Extension to the Power Model

PowerTimer, our power model, differs from existing academic microarchitectural performance simulators primarily in energy-model formation. The base energy-models are derived from circuit-level power analysis that has been performed on structures in a current, high-performance PowerPC processor. This analysis has been performed at the macro-level and in general multiple macros will combine to form microarchitectural level structures corresponding to units within our performance model. PowerTimer models over 60 microarchitectural structures which are defined by over 400 macro-level power equations.

2.2.1 Impact of Clock Gating Methodology on SMT

PowerTimer uses microarchitectural activity information from the Turandot model to scale down the unconstrained dynamic power under a variety of clock gating assumptions. In this study, we use a realistic form of clock gating which considers the applicability of clock gating on a per-macro basis to scale down the dynamic power depending on microarchitectural event counts. We determine which macros can be clock gated in a fine-grained manner (per-entry or per-stage clock gating) and which can be clock gated in a coarse-grained manner (the entire unit must be idle to be clock gated). For some macros (in particular control logic), we do not apply any clock gating; this corresponds to about 20-25% of the unconstrained dynamic power dissipation. Typically, the overall savings due to clock gating relative to the unconstrained dynamic power is roughly 40-50%. SMT machines tend to increase the utilization of the pipeline and thus the amount of power reduced by clock gating will decrease.

There are several styles of clock gating that we apply depending on the specific macro. These include valid and stall gating for latch-based structures and read and write port gating for array structures. Valid-bit clock gating is commonly used in pipeline latches and relatively small memory structures that are designed using latch-and-mux schemes (e.g. issue queues, instruction buffers, etc). In this style of gating, a valid-bit is associated with every bank of latches and the local clock buffer of the latch bank is gated when the valid-bit is not set. For array structures such as caches and large RAM-banks in certain queue structures, the array structure utilization is proportional to the number of read and write accesses to the structure.

SMT impacts the resource utilization within a microprocessor. This impact varies depending on the style of clock gating that exists in the underlying structures. For example, if a queue uses valid-bit based clock gating, and the occupancy rate of the queue increases, it is likely to see increases in its dynamic power dissipation. On the other hand, the impact of SMT on array structures may be small if the total number of accesses is roughly constant.

2.2.2 Modeling Increased Resource Needs for SMT

The majority of structures in a superscalar pipeline can be shared when augmenting the microprocessor for SMT. However, architected state must be duplicated for additional threads and new performance bottlenecks may arise requiring extension of shared resources. The major anticipated resource needs for SMT extensions can be categorized into the following.

- Resource Duplication. Structures such as the program counter must be duplicated for each thread. In this case we increase the power dissipation proportionally to the number of threads in the machine. Since only a very small portion of the resources need to be duplicated for a Power4-like architecture, the impact of increased power due to resource duplication is insignificant.
- Latch-based Queue Structures. Our analysis of latch-based queue structures determines that because the power dissipation is dominated by the latch and clocking circuitry, the power dissipation increases nearly linearly with the increase in the number of entries and bits per entry of these structures. We use the formula

$$Power_{new} = \frac{Entries_{new}}{Entries_{base}} * Power_{base} * PowerFactor \quad (1)$$

  Our default power model assumes that $PowerFactor$ is 1.0 (linear scaling) and we consider alternative values in our sensitivity analysis.
- Array Based Structures. For array-based structures, we utilize the empirical macro-level data from PowerTimer as our base value and scale this base value (for size and associativity) with estimates based on analytical models built into Wattch [3].

3. SIMULATION SETUP

3.1 Microarchitecture & Simulator

We use PowerTimer to model an out-of-order, superscalar processor with resource configuration similar to current generation microprocessors. The overall processor organization is shown in Figure 1 and Table 1 describes the configuration of our baseline processor for the single-threaded design point. As shown in Figure 1, the simulated processor can be logically divided into six major units: IFU, IDU, ISU, LSU, FXU, and FPU. The components of these units are listed below:

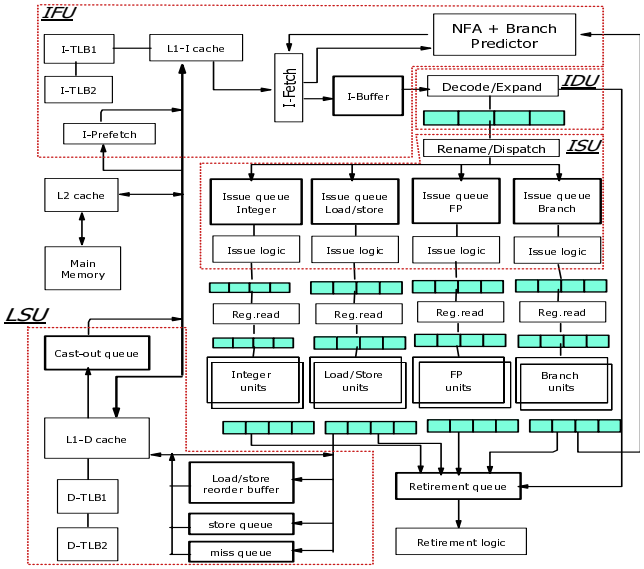

Figure 1: Modeled Processor Organization

Processor Core	
Dispatch Rate	5 instructions per cycle
Reservation stations	mem/fix queue (2x20), fpq (2x5)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BRU
Physical registers	80 GPR, 72 FPR
Branch predictor	16K-entry bimodal, 16K-entry gshare, 16K-entry selector, all with 1-bit entries
Memory Hierarchy	
L1 Dcache Size	32KB, 2-way, 128B blocks
L1 Icache Size	64KB, 2-way, 128B blocks
L2 I/D	1MB, 4-way LRU, 128B blocks 9-cycle latency
Memory Latency	77 cycles

Table 1: Configuration of simulated processor

- Instruction Fetch Unit (IFU): IFU includes program counters, level-one instruction cache, instruction TLBs, instruction buffer, branch predictor, next fetch address predictor (NFA), return address stack, etc.
- Instruction Decode Unit (IDU): IDU includes instruction decoder, microcode ROM, etc.
- Instruction Sequencing Unit (ISU): ISU includes register renamers, reservation stations, and retirement queue, etc.
- Load/Store Unit (LSU): LSU includes effective address calculator, level-one data cache, data TLBs, cast-out queue, load reorder buffer, store queue, load miss queue, etc.
- Fixed-point Execution Unit (FXU): FXU includes integer ALUs, integer multipliers/dividers, shifters, integer register file, etc.
- Floating-point Execution Unit (FPU): FPU includes floating-point pipelines, floating-point register file, etc.

3.2 Benchmark Pairs

For this study, we use 10 SPEC2000 integer benchmarks for our single thread experiments. They are compiled by *xlc* compiler with -O3 option. The static trace generation tool generates the final static traces by skipping the first 1B instructions and then tracing for 100M instructions in 50M instruction chunks, skipping 100M instructions between chunks.

We use pairs of single-thread benchmarks to form dual-thread SMT benchmarks. There are many possibilities for forming the pairs from these 10 benchmarks. We use the following methodology to form our pairs. First, we let each single thread benchmark combine with itself to form a pair, which gives us a total of 10 pairs. We then form several pairs by combining different benchmarks, after categorizing the benchmarks into four major categories: high IPC or low IPC, memory intensive or not memory intensive. We then form six pairs (gzip+perlbmk, gcc+gap, twolf+mcf, parser+bzip2, bzip2+twolf, gcc+mcf) of dual-thread benchmarks by selecting unique combinations of benchmarks with these categorizing criteria. We do not consider the operating system scheduling effect on SMT performance in this paper.

3.3 SMT Speedup Metric

Comparison of different SMT configurations, or comparison of an SMT configuration against a single-threaded configuration, is difficult. As Sazeides and Juan [10] have shown, IPC can be misleading unless exactly the same instruction count for each thread is used in all experiments. Otherwise, a high IPC may be achieved with a skewed load balance. Snavely et al. [12] also argue that SMT simulations should not be stopped when the first thread completes to perform comparison only on the portion of a workload that experiences multithreaded execution. This unfairly benefits SMT configurations by not accounting for periods of less than maximum throughput. When performing energy-efficiency studies, it also overlooks the impact of SMT energy overheads that are present even when only one thread is executing. Both groups propose similar metrics for computing an "SMT speedup". The goal is to distinguish between configurations that achieve high throughput at the expense of a single thread from those that do so with balanced throughput from both threads.

Sazeides and Juan propose that

$$\text{SMT speedup} = \frac{\sum L_{nonSMT}[i]}{L_{SMT}} \tag{2}$$

where L_i is the execution latency of the i'th thread on a single-threaded system, and L is the execution latency of the workload on an SMT system. A drawback to this mechanism is that L_{SMT} is determined by the thread that finishes last, and it cannot distinguish between different execution rates for other threads.

Snavely et al. propose that

$$\text{SMT speedup} = \sum \frac{IPC_{SMT}[i]}{IPC_{nonSMT}[i]} \tag{3}$$

where $IPC_{SMT}[i]$ is the IPC of just the i'th thread during an SMT execution and $IPC_{nonSMT}[i]$ is its IPC during single-threaded execution. This considers how each thread performs under SMT relative to its non-SMT performance, so we choose this metric for our speedup computations. All speedups are computed relative to the IPC of each workload on the baseline, non-SMT machine.

In contrast to evaluating performance, evaluating energy efficiency should use traditional, simple unweighted metrics. Total energy consumed during an experiment is the appropriate value to use for energy metrics, and simple end-to-end execution latency is the appropriate value to use for delay with energy-efficiency metrics like energy-delay^2. There are two reasons for this. First, unlike the tradeoff

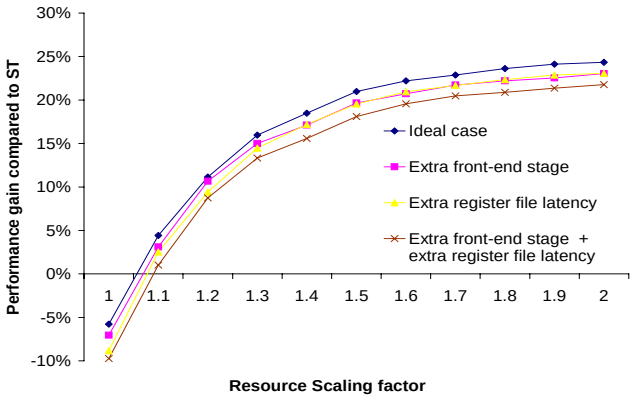

Figure 2: Performance of SMT vs. ST

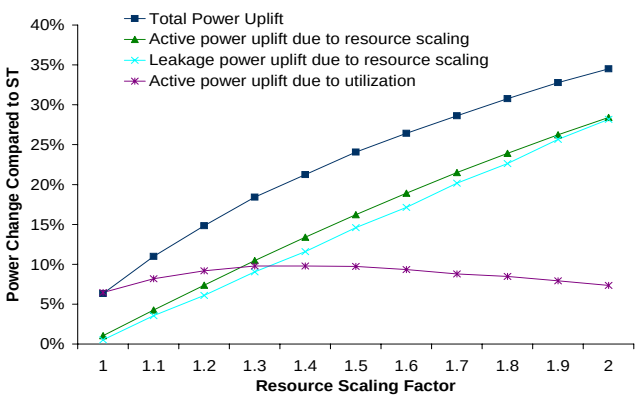

Figure 3: Power Dissipation of SMT vs. ST

between energy and execution speed, it is not clear how to trade off energy and load balance. Second, using weighted SMT speedup in an energy-efficiency metric could yield the counter-intuitive result that, among two SMT configurations with equal end-to-end execution latencies, a result with higher energy consumption is preferred.

4. RESULTS

In this section, we discuss the relative power-performance efficiency of SMT, analyze the relative impact of SMT power uplift factors, and discuss sensitivities to resource sizes, leakage power, and our power modeling methodology.

4.1 Power-performance Efficiency of SMT

To provide a balanced approach to support the increased number of inflight instructions provided by SMT, we perform a "ganged" scaling of all instruction buffers/queues, including instruction buffer, retirement queue, reservation stations, and physical registers. We use *resource scaling factor* to indicate the magnitude increase of queues, buffers, and physical register files compared to the base case shown in Table 1. A resource scaling factor of 1.0 corresponds to the case where these structures are sized the same as the base case, while a resource scaling factor of 2.0 means all the structures mentioned are double-sized. We do not scale the memory hierarchy and memory-related queues (including load reorder queue, store reorder queue, and load miss queue), because our sensitivity study indicates that their current sizes, as in the base case, do not constitute a performance bottleneck for the benchmarks we studied. Since only part of the resources are upscaled, we estimate the overall core area increase (excluding L2 cache) to be around 10% with a resource scaling factor of 1.5.

Figure 2 summarizes the performance benefit for SMT over the baseline single-threaded (ST) microprocessor, when varying the resource scaling factor. The numbers shown are the average performance for all the SMT pairs we simulated. The four curves correspond to different assumptions about the extra latencies SMT will incur, including the ideal SMT machine, where no extra latencies are added, a machine with an added pipestage in the front-end to account for thread selection logic, a machine with an added pipestage in the register file access to account for the larger register file, and finally, a machine that incurs both of the above latencies. We see from the figure that, at resource scaling factor of 1.0, thread contention is a serious problem, and even the ideal SMT machine suffers a 5% performance loss. The performance increases significantly when extra resources are added. At 1.5x scaling, the SMT performance benefit increases to around 21% for the ideal machine and 19% for the machine with both additional latencies. The curves begin to saturate with resource scaling factor of about 1.5, after which point increasing resources sees only diminishing performance gains. An interesting observation from the figures is that the four curves have very similar trends, indicating that the different latency assumptions do not change the SMT performance trend while varying resource scaling factor.

Now we look into the power dissipation of SMT. There are two major factors that cause SMT power uplift – the uplift due to resource duplication and resource sizing and the power uplift due to increased utilization (leading to reduced clock gating potential). PowerTimer allows us to measure the contributions of these two major components by providing power statistics with and without the power uplift applied by resource scaling. Figure 3 details the additional power dissipation that SMT incurs over the single-threaded machine and breaks down the two components of SMT power uplift. At the 1.5x scaling point the total core power has increased by 24% relative to the single-thread machine. The increase in processor utilization accounts for about 8% of this power increase and the remainder is due to the increased resource sizings. The power uplift due to processor utilization exhibits an interesting trend – with very small (1x) and very large (2x) values of resource scaling factor the power uplift is relatively small (5-6%). We explore this trend in more detail in Section 4.2 when we discuss the per-unit power uplift breakdown.

Figure 3 also breaks down the power uplift due to increased leakage power as we increase the size of resources. In our model, we estimate leakage power as a fraction of *unconstrained* active power and we do not scale leakage power with utilization. For our baseline model, we assume that this fraction is 0.1 and we call this variable *leakage factor*. We see that for the power uplift due to resource scaling, leakage power and active power track very closely. However, because the leakage power does not incur the additional power overhead due to increased utilization in the SMT machine, leakage power does not grow as quickly as active power. In

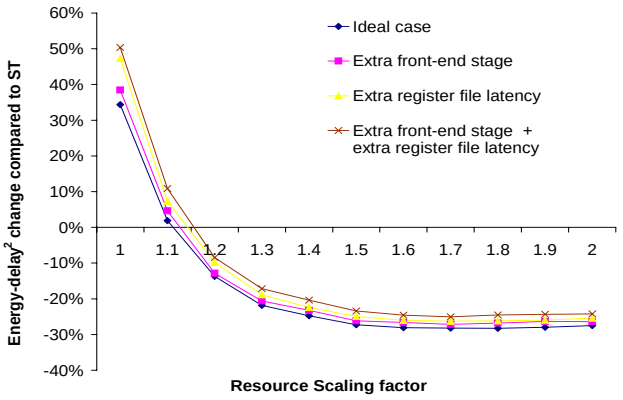

Figure 4: Energy-Delay^2 of SMT vs. ST

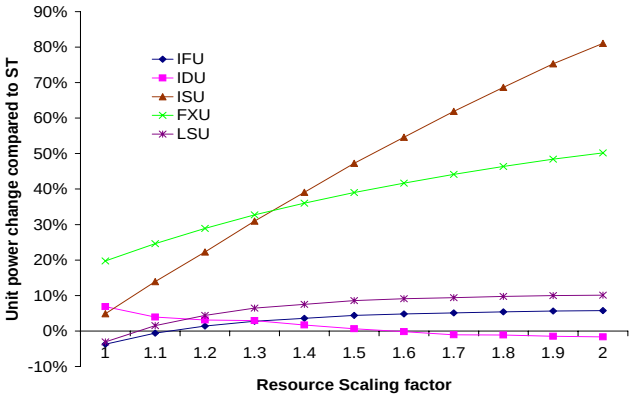

Figure 5: Power Dissipation Breakdown by Units

Section 4.3 we consider the sensitivity of our results to leakage factor.

For high-performance processors power-efficiency can best be quantified by the Energy-Delay^2 metric. Improvements in ED^2 correspond to power-performance efficiency benefits that exceed the cubic benefit derived by simply tuning the full-chip clock frequency and supply voltage [2]. Figure 4 provides the results for ED^2 and we can see that SMT is indeed very power-efficient and ED^2 is minimized with the SMT processor with 1.6x resource scaling. This is not surprising, given that SMT performance gain starts to saturate at around 1.5x, while power dissipation increases continue with larger values of resource scaling factor.

4.2 Breakdown of SMT power overheads by unit

We can obtain a better understanding of the power overheads associated with SMT by breaking down the power uplift by unit. Figure 5 shows the power increase under SMT for five major units within the microprocessor. The instruction sequencing unit (ISU) clearly stands out as experiencing the largest power changes, primarily because almost all of its subunits, such as reservation stations, register renamers, and retirement queue, are scaled to support SMT. The fixed-point execution unit (FXU) exhibits similar behavior, albeit milder, because the integer register file, which is also scaled under SMT, is in this unit. On the other hand, the power dissipation increase in the instruction fetch unit (IFU) and

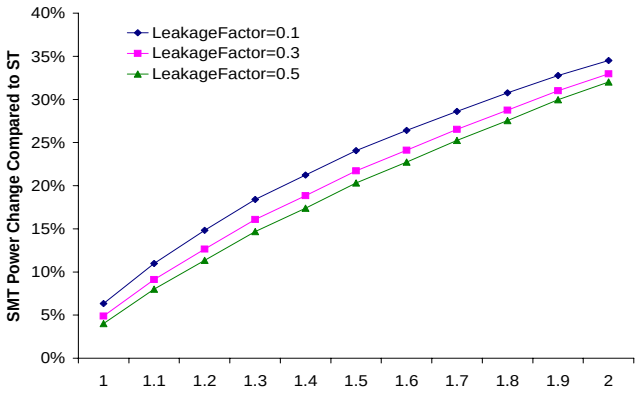

Figure 6: Impact of Leakage on Power Dissipation of SMT vs. ST

load/store unit (LSU) is primarily a result of increased utilization, as most of their components stay unchanged from ST to SMT. With more load/stores executed under SMT, at 1.5x resource scaling the LSU dissipates 10% additional power. The utilization uplift of both of these structures saturates when the larger instruction buffers and register files become large enough to support the ILP in both simultaneous threads.

The instruction decoding unit (IDU) displays behavior that is quite different than the other units. At the 1x scaling ratio, the power increase with SMT is roughly 10%, but this power delta gradually reduces as the resources increase. Our investigation reveals that at 1x scaling, the small instruction queues and physical register files are a severe performance bottleneck. This causes congestion within the IDU since the IDU decouples the IFU and ISU. Since the IDU utilizes valid-bit based clock-gating, the increased occupancy leads to higher power dissipation compared to the single-thread base case. As we upscale the resources in the ISU, the performance bottleneck at the ISU is gradually removed, reducing IDU power.

4.3 Sensitivity to Leakage Power

As process technologies migrate to smaller channel lengths and lower threshold voltages, static leakage currents become a major concern in the design of microprocessors. In this section, we consider the power overheads of SMT compared to single-thread architectures in technologies where leakage is a significant fraction of total power dissipation.

Figure 6 shows the total power increase (including active and leakage) of SMT compared to our single-thread baseline machine. We represent the future technologies by varying the leakage factor (LF) of our design from 0.1 (our baseline) to 0.5. As described in Section 4.1, we define leakage factor to be the fraction of the total unconstrained chip power that is leakage power.

Figure 6 shows that the total power uplift decreases slightly with leakier process technologies. At the 1.5x scaling point, the total power uplift is 24% with LF = 0.1, but reduces to 20% with LF = 0.5. This result is intuitive – as active power becomes a smaller fraction of the total power dissipation, the SMT machine's increase in utilization, and the corresponding reduction in clock gating potential, has less of an impact because clock gating only reduces active power.

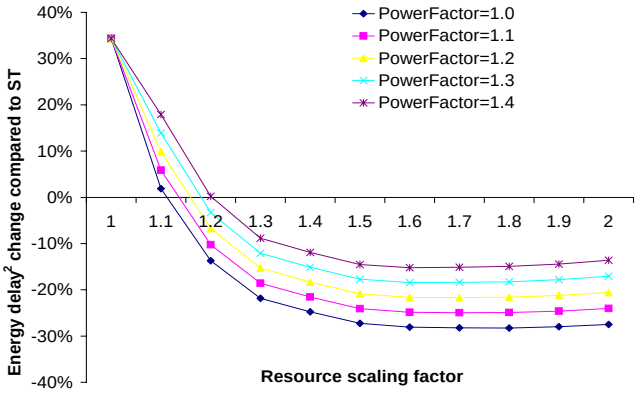

Figure 7: Impact of PowerFactor on Energy-Delay^2 of SMT vs. ST

4.4 Sensitivity to Resource Power Scaling

In this section we consider the impact of our scaling assumptions on our estimates for ED^2. For many of the structures, as we perform resource scaling, we assume that an increase in the number of entries has a linear increase in the unconstrained power dissipation, i.e. $PowerFactor = 1.0$ in Equation 1.

However, there may be cases where this assumption is too conservative. For example, the input and output (de)-multiplexors may become a more significant portion of power dissipation for large queue structures, and this could cause the power to grow super-linearly. Figure 7 shows the impact of varying the $PowerFactor$ variable in Equation 1 from 1x (linear scaling) to 1.4x. It is apparent that while the overall ED^2 savings will decrease considerably, the optimal design point is around 1.6x for all the $PowerFactor$ we considered. This indicates that a power model with a slightly inaccurate $PowerFactor$ can still have a meaningful projections for the trend for ED^2 of the SMT processor. This is encouraging, since for many architectural studies, relative accuracy is sufficient because early-stage architectural studies are primarily intended to narrow the focus of design choices. Later studies after design implementation begins can provide more detailed models that improve absolute accuracy.

5. FUTURE WORK AND CONCLUSIONS

Our future work seeks to further validate our performance and power extensions for SMT. The baseline single-threaded performance model has been extensively validated against a pre-RTL, latch-accurate processor model for a current generation microprocessor [8]. For the SMT extension, we will focus on validating the two major perturbations caused by SMT: increased utilization and resource scaling. Our strategy to validate utilization is as follows: we will construct simple microbenchmarks, which are mainly loop-like kernels, whose resource utilization can easily be deduced under single-thread or SMT environment. We run such microbenchmarks, collect the utilization, and compare them with our offline calculation to make sure they match each other. In our next step, we plan to validate the SMT performance model against the product-level processor model for the most recent IBM processor.

We also seek to further understand the power-performance impact of exploiting on-chip thread-level parallelism including tradeoffs for exploiting this parallelism with multiple cores. We also plan to analyze the worst-case temperature behavior of key structures within the microprocessor, in particular within the ISU, which incurs a sharp increase in power dissipation with SMT.

This paper describes the modeling extension and validation strategy to study the power-performance impact of SMT. We have also performed a detailed design space study of the impact of augmenting an existing POWER4-like microarchitecture with SMT. Overall, we conclude that SMT is a power-efficient design paradigm for modern, superscalar microarchitectures. After careful resource size tuning within the processor core, designers can expect performance gains of nearly 20% with a power uplift of roughly 24% leading to significant reduction in ED^2.

6. REFERENCES

[1] D. Brooks, P. Bose, V. Srinivasan, M. Gschwind, P. G. Emma, and M. G. Rosenfield. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal of R&D*, 47(5/6), 2003.

[2] D. Brooks et al. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *27th International Symposium on Computer Architecture (ISCA)*, 2000.

[4] J. Burns and J.-L. Gaudiot. SMT layout overhead and scalability. *IEEE Trans. Parallel Distrib. Syst.*, 13(2):142–155, 2002.

[5] R. Kalla, B. Sinharoy, and J. Tendler. POWER5: IBM's next generation power microprocessor. In *Proc. 15th Hot Chips Symp*, pages 292–303, August 2003.

[6] D. Koufaty and D. T. Marr. Hyperthreading technology in the netburst microarchitecture. *IEEE Micro*, 23(2):56–65, 2003.

[7] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton. Hyper-threading technology architecture and microarchitecture. *Intel Technology Journal*, 6(1):4–15, Feb. 2002.

[8] M. Moudgill, P. Bose, and J. H. Moreno. Validation of Turandot, a fast processor model for microarchitecture exploration. In *Proceedings of IEEE Internatioanl Performance, Computing and Communications Conference*, pages 451–457, February 1999.

[9] M. Moudgill, J.-D. Wellman, and J. H. Moreno. Environment for PowerPC microarchitecture exploration. *IEEE Micro*, 19(3):15–25, 1999.

[10] Y. Sazeides and T. Juan. How to compare the performance of two SMT microarchitectures. In *IEEE International Symposium on Performance Analysis of Systems and Software*, November 2001.

[11] J. Seng, D. Tullsen, and G. Cai. Power-sensitive multithreaded architecture. In *Proceedings of the 2000 International Conference on Computer Design*, pages 199–208, 2000.

[12] A. Snavely and D. M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Ninth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 234–244, November 2000.

[13] D. M. Tullsen, S. Eggers, and H. M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, 1995.