

Power-Sensitive Multithreaded Architecture

John S. Seng

Dean M. Tullsen

*George Z.N. Cai

Dept. of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0114

*Intel Corporation
RA2-401
2501 N.W. 229th Ave
Hillsboro, OR 97124

Abstract

The power consumption of microprocessors is becoming increasingly important in design decisions, not only in mobile processors, but also now in high-performance processors. Power-conscious design must therefore go beyond technology and low-level design, but also change the way modern processors are architected.

A multithreading processor is attractive in the context of low-power or power-constrained devices for many of the same reasons that enable its high throughput. Primarily, it supplies extra parallelism via multiple threads, allowing the processor to rely much less heavily on speculation. We show that a simultaneous multithreading processor utilizes up to 22% less energy per instruction than a single-threaded architecture. We also explore other power optimizations that are particular to multithreaded architectures, either because they are unavailable to or unreasonable for single-thread architectures.

1. Introduction

Hardware multithreading is poised to begin appearing in commodity processors just as concerns about the energy and power dissipation of modern CPUs are becoming more critical. In this paper, we demonstrate that (1) multithreading is an inherently energy efficient architecture and (2) a simultaneous multithreaded architecture allows power/performance design tradeoffs that are not available or reasonable in a single-threaded machine.

Processor power and energy consumption are of concern in two different operating environments. The first is that of a mobile computer, where battery life is still very limited. While the overall energy consumption of a microprocessor is being reduced because of voltage scaling, dynamic clock frequency reduction, and low power circuit design, optimizations can be applied at the architecture level as well.

The other environment where power consumption is important is that of high performance processors. These pro-

cessors are used in environments where energy supply is not typically a limitation. For high performance computing, clock frequencies continue to increase, causing the power dissipated to reach the thresholds of current packaging technology. When the maximum power dissipation becomes a critical design constraint, that architecture which maximizes the performance/power ratio thereby maximizes performance.

Multithreading [15, 14, 1] is a processor architecture technique which has been shown to provide significant performance advantage over conventional architectures which can only follow a single stream of execution. Simultaneous multithreading (SMT) [15, 14] can provide up to twice the throughput of a dynamic superscalar single-threaded processor. Announced architectures that will feature multithreading include the Compaq Alpha EV8 [5] and the Sun MAJC Java processor [12], joining the existing Tera MTA supercomputer architecture [1].

In this paper, we show that a multithreading processor is attractive in the context of low-power or power-constrained devices for many of the same reasons that enable its high throughput. First, it supplies extra parallelism via multiple threads, allowing the processor to rely much less heavily on speculation; thus, it wastes fewer resources on speculative, never-committed instructions. Second, it provides both higher and more even parallelism over time when running multiple threads, wasting less power on underutilized execution resources. A multithreading architecture also allows different design decisions than are available in a single-threaded processor, such as the ability to impact power through the thread selection mechanism.

This paper is organized as follows. Section 2 discusses related studies. Section 3 describes the simulator and power model used in this study. Section 4 describes the methodology used. Section 5 presents the baseline power consumption of a multithreaded processor. Section 6 presents power optimizations specifically targeted to a multithreaded processor. Section 7 concludes.

2. Related Work

This section describes other architecture-level power models, as well as related studies on power and energy conservation, speculation control and multithreading.

Recent architecture level models include [2] and [16]. Chen, et al. discuss an architecture level power model in [4] that was verified against a commercial RISC processor and DSP engine. They discuss the performance of the power model when running simple synthetic benchmarks.

Cai, et al. [3] describe a power model that is most similar to the model utilized in our research. The power model described in that paper is added to SimpleScalar. The power model we use is combined with the SMTSIM simulator [13] and is scaled to model a multithreaded processor. For our power control results we looked at power consumption over a fixed interval period and we also use the power model for average power consumption analysis.

Pipeline gating [11] uses branch confidence prediction [9, 7] to control speculation in a convention single-threaded pipeline, stopping fetch beyond low-confidence branches. We apply somewhat related techniques to both single-thread and multithreaded execution. Unlike that paper, we have a power model to evaluate its effectiveness, and exploit some more interesting opportunities presented by multithreading.

Horowitz et al. [6] perform a study on general microprocessors and the power consumed by various implementation features of the processors. That paper examines the power effects of pipelining and superscalar issue, but does not consider the effects of multithreading.

Simultaneous multithreading has been shown to be an effective architecture to increase processor throughput both in multiprogrammed [15, 14] and parallel execution [10]. Previous work has demonstrated SMT's reduced dependence on speculation to achieve parallelism [14, 8].

3. Modelling Power

This section describes the power model used to produce the energy and power results in the rest of the paper. This power model is integrated into a detailed cycle-by-cycle instruction-level architectural simulator, allowing the model to accurately model both useful and non-useful (incorrectly speculated) use of all processor resources.

The power model utilized for this study is an area-based model, similar in that respect to [3]. In this power model, the microprocessor is divided into several high-level units, and the corresponding activity of each unit is used in the computation of the overall power consumption. The following processor units are modeled: L1 instruction cache, L1 data cache, L2 unified cache, fetch unit, integer and floating point instruction queues, branch predictor, instruction

TLB, data TLB, load-store queue, integer and floating-point functional units, register file, register renamer, completion queue, and return stack.

The total processor energy consumption is the summation of the unit energies where each unit energy is equal to:

$$\text{energy per unit} = \text{unit activity factor}(s) * \text{unit area} * \text{energy density}$$

An activity factor is a defined statistic measuring how many architectural events a program or workload generates for a particular unit. For a given unit, there may be multiple activity factors each representing a different action that can occur within the unit. Activity factors represent high level actions and therefore are independent of the data values causing a particular unit to become active. We can compute the overall power consumption of the processor on a cycle-by-cycle basis by summing the energy usage of all units in the processor.

The entire power model consists of 44 activity factors. Each of these activity factors is a measurement of the activity of a particular unit or function of the processor (e.g. number of instruction queue writes, L2 cache reads). The model does not assume that a given factor or combination of factors exercises a microprocessor unit in its entirety, but instead is modeled as exercising a fraction of the area depending on the particular activity factor or combination of activity factors. The weight of how much of the unit is exercised by a particular activity is an estimate based on knowledge of the unit's functionality and assumed implementation.

Each unit is also assigned a relative area value which is an estimate of the unit's size given its parameters. In addition to the overall area of the unit, each unit is modeled as consisting of a particular ratio of 4 different circuit types: dynamic logic, static logic, memory, and clock circuitry. Each circuit type is given an energy density. These energy density factors are similar to the ones described in [3]. The ratio of circuit types for a given logic unit are estimates based on engineering experience and consultation with processor designers.

The advantage of an area-based design is its adaptability. It can model an aggregate of several existing processors, as long as average area breakdowns can be derived for each of the units. And it can be adapted for future processors for which no circuit implementations exist, as long as an estimate of the area expansion can be made. In both cases, we rely on the somewhat coarse assumption that the general circuit makeup of these units remains fairly constant across designs. This assumption should be relatively valid for the types of high-level architectural comparisons made in this paper, but might not be for more low-level design options.

Our power model is an architecture-level power model, and is not intended to produce precise estimates of absolute whole-processor power consumption. For example, it

does not include dynamic leakage factors, I/O power (e.g., pin drivers), etc. However, the power model is useful for obtaining relative power numbers for comparison against results obtained from the same power model. The numbers that are obtained from the power model are independent of clock frequency (again, because we focus on relative values).

The relative area estimates for the various processor components are based on actual scale die photographs of several PowerPC, MIPS, and Intel processors. In order to approximate the area overhead of a multithreaded processor as compared to a single-threaded superscalar microprocessor, the areas of the components which are critical to multithreading have been scaled by 10%. The area estimate for the larger physical register file required by multithreading has been scaled up by 40%. Other components (e.g., functional units, caches, etc.) are also scaled appropriately to account for our baseline machine model, which is larger than the existing processors studied.

Because our power model is integrated into the cycle-by-cycle architecture simulator rather than calculated from the final results of the simulator, it can be used for the identification of power spikes or temporary hot spots. For this paper we focus mostly on average results over the entire simulation run; however, it does allow us to model fine-grain power feedback mechanisms.

4. Methodology

We use several metrics to understand the tradeoffs of various designs. From a performance perspective, we look at (committed) instructions per cycle (IPC). As for power metrics, we look at two different ones. In order to gauge operating efficiency, we are concerned about the energy used per unit of work. In particular, we measure the average energy used for each useful (committed) instruction that goes through the processor. Since an application can be expected to commit a constant number of instructions, this is equivalent (as a metric) to the energy used per program. For a high performance processor, we also want to know the power consumed by the processor when running a particular application - the numbers we present are for average power, but we discuss peak power when relevant.

When simulating multithreaded execution over short periods, it is critical to maintain a constant contribution from each thread. That is, a technique that favors a high-IPC or low-power thread systematically would show improvement over the simulation run simply due to an artificially increased contribution from the favored thread. This is not an improvement that would be seen in a real system (which would eventually have to run the non-favored thread). For this reason we run a constant number of instructions for *each* benchmark for each run. Therefore, each multi-

Benchmark	input	Insts (millions)
gcc	cp-decl.i	350
go	5stone21	402
ijpeg	vigo.ppm	670
li	ref	460
m88k	ref	434
perl	scrabbl	378

Table 1. The benchmarks used in this study.

Parameter	Value
Fetch bandwidth	2 threads 8 instructions total
Functional Units	3 FP, 6 Int (4 load/store)
Instruction Queues	32-entry FP, 32-entry Int
Inst Cache	64KB, 2-way, 64-byte lines
Data Cache	64KB, 2-way, 64-byte lines
L2 Cache (on-chip)	1 MB, 4-way, 64-byte lines
L3 Cache (off-chip)	4 MB
Latency (to CPU)	L2 6 cycles, L3 18 cycles, Memory 80 cycles
Pipeline depth	9 stages
Min branch penalty	7 cycles
Branch predictor	4K gshare
Instruction Latency	Based on Alpha 21164

Table 2. The processor configuration.

threaded simulation run finishes threads at different times, eventually finishing the simulation with only a single thread running. The total number of instructions executed for each benchmark is given in Table 1.

The benchmarks all come from the SPEC 95 integer suite, and all the inputs come from the reference set. We run only single-thread versions of the benchmarks. Multithreading is achieved by running a different program in each hardware context.

The baseline processor simulated is an 8-issue simultaneous multithreading superscalar processor configured as specified in Table 2.

Execution is simulated on the SMTSIM simulator [13] which runs unaltered Alpha executables. The simulator already contains full support for accurate emulation of wrong-path execution following a mispredicted branch prediction. The architectural simulator is enhanced to support the various power optimizations in this paper. Integrated into the SMTSIM simulator is the cycle-by-cycle power simulator, as detailed in Section 3.

In general, we have *not* accounted for inherent hardware differences between a single-threaded processor and a multithreaded processor in these results. Although our power model easily handles some of the differences (e.g., the larger register file), it would certainly not handle all (e.g., increased control logic distributed throughout the processor). Therefore, we adhere to a single model aimed at the multithreaded implementation. The results shown, therefore, more accurately compare the two *execution modes* rather than the hardware architectures.

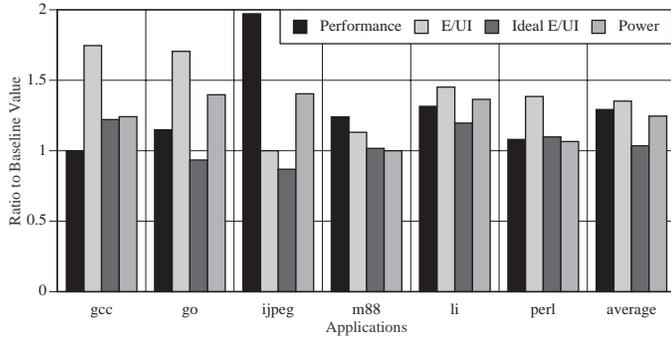


Figure 1. The relative performance, energy efficiency, and power results for the six benchmarks when run alone. All results are normalized to a baseline.

5. Power Consumption of a Multithreaded Architecture

This section establishes the power and energy characteristics of multithreaded execution. It examines performance (IPC), energy efficiency (energy per useful instruction executed, E/UI), and power (the average power utilized during each simulation run) for both single-thread and multithread execution.

All results (including IPC) are normalized to a baseline (in each case, the lowest single-thread value). This is done for two reasons — first, the power model is not intended to be an indicator of absolute power or energy, so this allows us to focus on the relative values; second, it allows us to view the diverse metrics on a single graph. Later results are normalized to other baseline values.

The single-thread results (Figure 1) show diversity in almost all metrics. Performance and power appear to be positively correlated; the more of the processor used, the more power used. However, performance and energy per useful instruction are somewhat negatively correlated; the fewer unused resources and the fewer wastefully used resources, the more efficient the processor.

Also included in the graph (the Ideal E/UI bar) is the energy efficiency of each application with perfect branch prediction. This gives an indication of the energy lost to incorrect speculation (when compared with the E/UI result). *Gcc* and *go* suffer most from low branch prediction accuracy. *Gcc*, for example, only commits 39% of fetched instructions and 56% of executed instructions. In cases such as *gcc* and *go*, the energy effects of mispredicted branches is quite large (35% - 40%).

Figure 2 shows that multithreaded execution has significantly improved energy efficiency over conventional processor execution. Multithreading attacks the two primary sources of wasted energy/power — unutilized resources, and most importantly, wasted resources due to incorrect

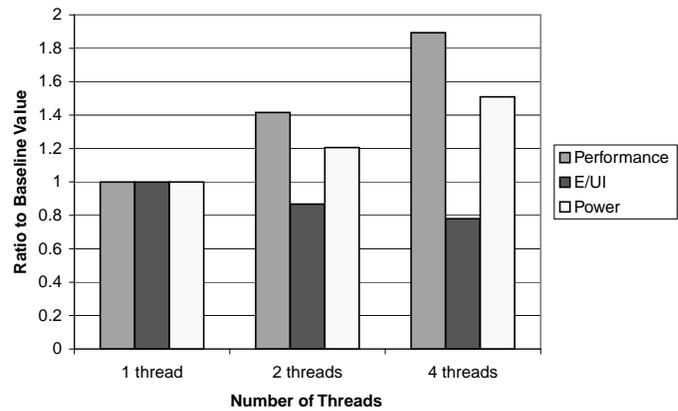


Figure 2. The relative performance, energy efficiency, and power results as the number of threads varies.

speculation. Multithreaded processors rely far less on speculation to achieve high throughput than conventional processors [14, 8].

With multithreading, then, we achieve as much as a 90% increase in performance (the 4-thread result), while actually *decreasing* the energy dissipation per useful instruction of a given workload. This is achieved via a relatively modest increase in power. The increase in power occurs because of the overall increase in utilization and throughput of the processor - that is, because the processor completes the same work in a shorter period of time.

Figure 3 shows the effect of the reduced misspeculation on the individual components of the power model. The greatest reductions come in the front of the pipeline (e.g. fetch), which is always the slowest to recover from a branch misprediction. The reduction in energy is achieved despite an increase in L2 cache utilization and power. Multithreading, particularly a multiprogrammed workload, reduces the locality of accesses to the cache, resulting in more cache misses; however, this has a bigger impact on L2 power than L1. The L1 caches see more cache fills, while the L2 sees more accesses per instruction. The increase in L1 cache fills is also mitigated by the same overall efficiency gains, as it sees fewer speculative accesses.

The power advantage of simultaneous multithreading can provide a significant gain in both the mobile and high performance domains. The E/UI values demonstrate that a multithreaded processor operates more efficiently, which is desirable in a mobile computing environment. For a high-performance processor, the constraint is more likely to be average power or peak power. SMT can make better use of a given average power budget, but it really shines in being able to take greater advantage of a peak power constraint. That is, an SMT processor will achieve sustained performance that is much closer to the peak performance (power)

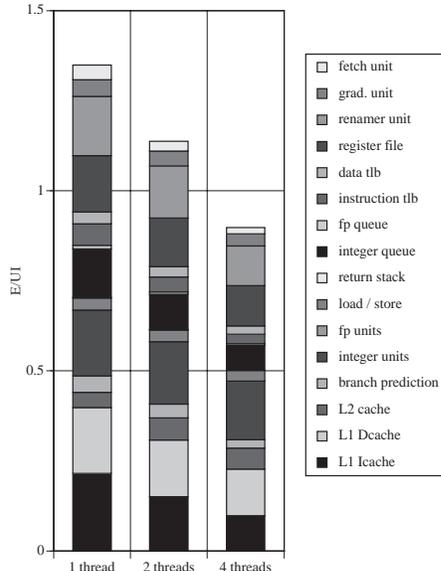


Figure 3. The contributors to overall energy efficiency for different thread configurations.

the processor was designed for than a single threaded processor. This is explored more closely in Section 6.2.

6. Power Optimizations

In this section, we examine power optimizations that will either reduce the overall energy usage of a multithreading processor or will reduce the power consumption of the processor. We examine the following possible power optimizations, each proving practical in some way by multithreading: reduced execution bandwidth, dynamic power consumption controls, and thread fetch optimizations.

Reduced execution bandwidth exploits the higher execution efficiency of multithreading to achieve higher performance while maintain moderate power consumption. Dynamic power consumption controls allow a high-performance processor to gracefully degrade activity when power consumption thresholds are exceeded. The thread fetch optimization looks to achieve better processor resource utilization via thread selection algorithms.

6.1. Reduced Execution Bandwidth

Prior studies have shown that an SMT processor has the potential to achieve double the instruction throughput of a single-threaded processor with similar execution resources [14]. This implies that a given performance goal can be met with a less aggressive architecture, possibly consuming less power. Therefore, a multi-threaded processor with a smaller execution bandwidth may achieve comparable performance to a more aggressive single-threaded pro-

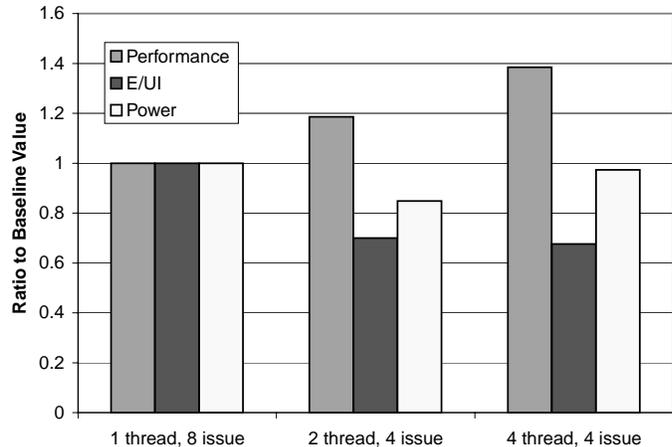


Figure 4. The performance, energy and power characteristics of a single-threaded 8-issue processor, compared to a multithreaded 4-issue processor.

cessor while consuming less power. In this section, for example, we compare the power consumption and energy efficiency of an 8-issue single-threaded processor with a 4-issue multithreaded processor.

We altered the simulation model to account for the power and architectural differences between the two architectures. The estimated sizes of the fetch unit and the register renamer were reduced by 50%. The estimated size of the issue queues were reduced by 25%. Although a complete scan of the issue queues is required, the maximum number of instructions that are released from the queues per cycle is cut in half. We also reduced the area of the retirement unit by 50% since half as many instructions are retired per cycle in the reduced bandwidth processor.

These results are shown in Figure 4. The 4-issue machine provides comparable and even greater performance than the 8-issue because of the performance enhancement provided by multithreading. When multithreading is enabled, the overall efficiency of the processor improves significantly, as can be seen by the reduction in energy per useful instruction. The power consumption also drops. We can see a slight increase in the power consumption when the multithreading level increases from 2 to 4 threads.

This shows that with a less aggressive machine, we can meet both higher performance and lower energy standards. In addition, we also do it with lower average power and much lower peak power (not shown, but the peak power will be dominated by the maximum IPC, which is halved). In a mobile environment, in particular, this can represent a clear win.

While probably less desirable for high-performance applications, this optimization does effectively target particularly power-intensive areas of the processor. The instruction queues and fetch unit are very active areas of the proces-

sor, and may very well represent hot spots that exceed local power dissipation constraints.

6.2. Controlling Power Consumption via Feedback

The previous section showed that we can meet a given performance goal while reducing energy, a desirable achievement for a mobile operating environment. This section strives to reduce the peak power dissipation while still maximizing performance, a goal that is likely to face future high-performance architects. In this section, the processor is given feedback regarding power dissipation in order to limit its activity.

Such a feedback mechanism does indeed provide reduced average power, but the real attraction of this technique is the ability to reduce peak power to an arbitrary level. A feedback mechanism that guaranteed that the actual power consumption did not exceed a threshold could make peak and achieved power consumption arbitrarily close. Thus, a processor could still be 8-issue (because that width is useful some fraction of the time), but might still be designed knowing that the peak power corresponded to 5 IPC sustained, as long as there is some mechanism in the processor that can guarantee sustained power does not exceed that level.

This section models a mechanism, applied to an SMT processor, which utilizes the power consumption of the processor as the feedback input. For this optimization, we would like to set an average power consumption target and have the processor stay within a reasonable tolerance of the target while achieving higher performance than the single threaded processor could achieve. Our target power, in this case, is exactly the average power of all the benchmarks running on the single-threaded processor. This is a somewhat arbitrary threshold, but makes it easier to interpret the results. The feedback is an estimated value for the current power the processor is utilizing. An implementation of this mechanism could use either on-die sensors or a 'power' value which is computed from some performance counters. The advantage of using performance counters is that the lag time for the value is much less than those obtained from a physical temperature sensor. For the experiments listed here, the power value used is the number obtained from the simulator power model. We used a power sampling interval of 5 cycles (this is basically the delay for when the feedback information can be used by the processor).

We show results for a variety of power control policies that are potentially beneficial on a multithreaded processor. In each of the techniques we define thresholds whereby if the power consumption exceeds a given threshold, fetching or branch prediction activity will be modified to curb power consumption. The threshold values are fixed throughout a given simulation. In all cases, fetching is halted for all

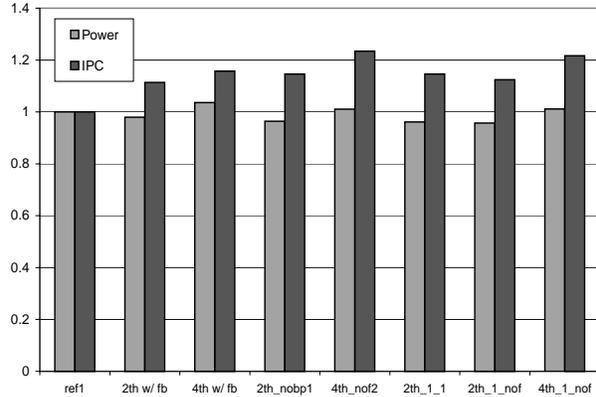


Figure 5. The average over all benchmarks of performance and power of 2 and 4 thread runs with feedback

threads when power consumption reaches 110% of the desired power value.

The *2th w/ fb* and *4th w/ fb* are 2 and 4 thread techniques which utilize 2 threshold values: one which is 10% below (first threshold) and another which is 10% above (second threshold) the average of the average power of the benchmarks running on a single threaded processor. Once the power consumption exceeds the lower threshold, the branch predictor is turned off (instructions are not fetched past unresolved branches) and the processor continues executing without speculation. If the power consumption continues to exceed the first threshold and also exceeds the second threshold, the fetch unit is stopped and does not resume fetching (still without branch prediction) until the power consumption drops below the second threshold. Once the power consumption drops below the first threshold, the branch predictor is turned back on.

The *2th_nobp1* involves simulations with 2 threads running where the branch prediction is stopped (for 1 of the threads) when power consumption reaches 90% of desired power. The *4th_nof2* demonstrates the effect of halting fetching for 2 threads out of 4 when power consumption reaches 90% of the desired power value.

The *2th_1_1* technique halts branch prediction for individual threads at various thresholds. When the processor power consumption reaches the first threshold (90% of the desired power value), branch prediction for the lower IPC thread is stopped. Next, branch prediction is stopped for both threads when power consumption equals 100% of the desired power value. Finally, fetching stops for both threads at 110% of the desired power value.

The *2th_1_nof* is a 2 thread mechanism where fetching halts for one of the threads when power reaches 90% of the desired power value. The *4th_1_nof* is similar but applies to 4 thread simulations. In these experiments, a thread at a

time stops fetching when power consumption reaches 80%, 90%, and 100% of the desired value.

Figure 5 show the performance and average power results of 1 thread running with full branch prediction (the reference case) and the results of the different power control mechanisms. The numbers represent the average over all the benchmarks for a given configuration. The results of the *2th w/ fb* mechanism is a performance improvement of 11% over the 1 thread result with branch prediction turned on all the time. The power results of these two experiments is approximately the same, with the processor with the feedback mechanism actually utilizing less power. The *4th w/ fb* results show a 15% improvement in performance with only a 3% increase in average power when compared against the standard 1 thread result. The other mechanisms show promise as well in limiting the power consumption. These results show that even a very tight power threshold feedback mechanism allows an SMT processor to provide higher performance.

Power feedback control is a technique which enables the designer to lower the peak power constraints on the processor. This technique is particularly effective in a multithreading environment for two reasons. First, even with the drastic step of eliminating speculation, even for fetch, an SMT processor can still make much better progress than a single-threaded processor. Second, we have more dimensions on which to scale back execution — the results were best when we took advantage of the opportunity to scale back threads incrementally.

6.3. Thread Selection

This optimization examines the effect of thread selection algorithms on power and performance. The ability to select from among multiple threads provides the opportunity to optimize for power consumption and operating efficiency when making thread fetch decisions.

The optimization we model involves two threads that are selected each cycle to attempt to fetch instructions from the I-cache. The heuristic used to select which threads fetch can have a large impact on performance [14]. This mechanism can also impact power if we use it to bias against the most speculative threads.

This heuristic does not, in fact, favor low-power threads over high-power threads, because that only delays the running of the high-power threads. Rather, it favors less speculative threads over more speculative threads. This works because the threads that get stalled become less speculative over time (as branches are resolved in the processor), and quickly become good candidates for fetch.

We modify the ICOUNT thread selection scheme, from [14], by adding a branch confidence metric to thread fetch priority. This could be viewed as a derivative of

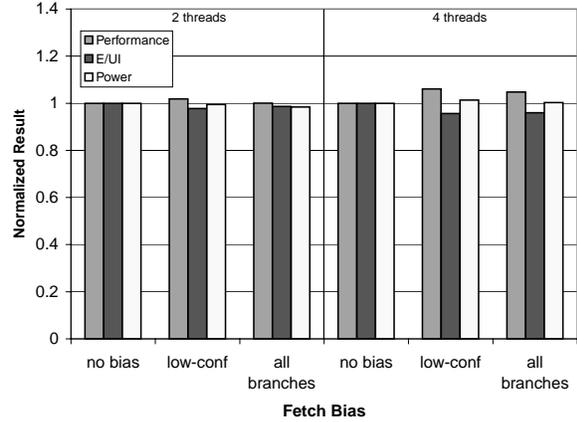


Figure 6. The performance, energy and power effects of using branch status to direct thread selection.

pipeline gating [11] applied to a multithreaded processor; however, we use it to change the fetch decision, not to stop fetching altogether. The *low-conf* scheme biases heavily against threads with more unresolved low-confidence branches, while *all branches* biases against the threads with more branches overall in the processor (regardless of confidence). Both use ICOUNT as the secondary metric. *all branches* is thus a hybrid of the ICOUNT and BRANCH fetching schemes in [14].

The confidence scheme used is a one-level dynamic confidence predictor as described in [9]. We used an xor indexing method into a 1K x 16 CIR history table. For the reduction function, saturating counters were used, with a threshold of 3.

Figure 6 shows that this mechanism has the potential to improve both raw performance and energy efficiency at the same time, particularly with the confidence predictor. For the 4 thread simulations, performance increased by 6.3% and 4.9% for the *low-conf* and *all branches* schemes, respectively. The efficiency gains should be enough to outweigh the additional power required for the confidence counters (not modeled), assuming the architecture did not already need them for other purposes. If not, the technique without the confidence counters was equally effective at improving power efficiency, lagging only slightly in performance.

This figure shows an improvement even with two threads, because when two threads are chosen for fetch in a cycle, one is still chosen to have higher priority and possibly consume more of the available fetch bandwidth [14].

This section has shown that a multithreading processor offers several interesting design optimizations, depending on where in the power-constrained design space the processor is targeted.

7. Conclusions

Microprocessor power dissipation is becoming increasingly critical, due to various pressures. Low-power embedded and mobile devices are increasing rapidly. Every generation of processor puts far greater demands on power and cooling than the previous. We are approaching a technological window when power may become a bottleneck before transistor count, even for high-performance processors. In that scenario the processor architecture that optimizes the performance/power ratio thereby optimizes performance.

In this paper we demonstrate that simultaneous multithreading is an attractive architecture when energy and power are constrained. It, in particular, can use significantly less power per instruction (and thus, per program) when multiple threads are in execution. It does so by providing more even parallelism, fewer underutilized resources, and significantly fewer wasted (due to incorrect speculation) resources.

Multithreading also provides other design options, as well. A given performance goal can be met while decreasing energy through the use of a reduced width multithreaded processor. Through the use of a feedback mechanism, actual power consumption can be arbitrarily close to the designed peak power, and do so over a much wider performance range than could a single-thread processor. Various thread selection mechanisms provide the means to even further reduce the amount of power wasted on misspeculated execution.

8. Acknowledgments

We would like to thank the anonymous reviewers for their useful comments. This work was funded by NSF CAREER grant No. MIP-9701708, an internship at Intel, and equipment grants from Compaq Computer Corporation.

References

- [1] R. Alverson, D. Callahan, D. Cummings, B. Koblenz, A. Porterfield, and B. Smith. The tera computer system. In *International Conference on Supercomputing*, pages 1–6, June 1990.
- [2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *27th Annual International Symposium on Computer Architecture*, June 2000.
- [3] G. Cai and C. H. Lim. Architectural level power/performance optimization and dynamic power estimation. In *Proceedings of Cool Chips Tutorial at 32nd International Symposium on Microarchitecture*, November 1999.
- [4] R. Chen, M. Irwin, and R. Bajwa. An architectural level power estimator. Dept. CSE - Pennsylvania State University, June 1998.
- [5] J.S. Emer. Simultaneous multithreading: Multiplying alpha's performance. In *Microprocessor Forum*, October 1999.
- [6] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. In *IEEE International Symposium on Low Power Electronics 1995*, October 1995.
- [7] D. Grunwald, A. Klauser, S. Manne, and A. Pleskun. Confidence estimation for speculation control. In *25th Annual International Symposium on Computer Architecture*, June 1998.
- [8] S. Hily and A. Sez nec. Out-of-order execution may not be cost-effective on processors featuring simultaneous multithreading. In *Proceedings of the Fifth International Symposium on High-Performance Computer Architecture*, pages 44–53, January 1999.
- [9] E. Jacobsen, E. Rotenberg, and J.E. Smith. Assigning confidence to conditional branch predictions. In *29th International Symposium on Microarchitecture*, December 1996.
- [10] J.L. Lo, S.J. Eggers, J.S. Emer, H.M. Levy, S.S. Parekh, R.L. Stamm, and D.M. Tullsen. Converting thread-level parallelism into instruction-level parallelism via simultaneous multithreading. *ACM Transactions on Computer Systems*, August 1997.
- [11] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: Speculation control for energy reduction. In *25th Annual International Symposium on Computer Architecture*, June 1998.
- [12] MAJC gives VLIW a new twist. *Microprocessor Report*, 13(12), September 1999.
- [13] D.M. Tullsen. Simulation and modeling of a simultaneous multithreading processor. In *22nd Annual Computer Measurement Group Conference*, December 1996.
- [14] D.M. Tullsen, S.J. Eggers, J.S. Emer, H.M. Levy, J.L. Lo, and R.L. Stamm. Exploiting choice: Instruction fetch and issue on an implementable simultaneous multithreading processor. In *23rd Annual International Symposium on Computer Architecture*, May 1996.
- [15] D.M. Tullsen, S.J. Eggers, and H.M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *22nd Annual International Symposium on Computer Architecture*, pages 392–403, June 1995.
- [16] N. Vijaykrishnan, M. Kandemir, M.J. Irwin, H.S. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using simplepower. In *27th Annual International Symposium on Computer Architecture*, June 2000.