Leveraging Simultaneous Multithreading for Adaptive Thermal Control

James Donald and Margaret Martonosi
Department of Electrical Engineering
Princeton University
{jdonald, mrm}@princeton.edu

Abstract

The continual increase in microprocessor transistor densities has led to major challenges in on-chip temperature management. Examining how emerging architectural paradigms scale from a thermal-aware design perspective is critical for sustaining high-performance computing. In this paper we explore a novel dynamic thermal management technique for simultaneous multithreaded processors. Unlike prior studies, rather than testing general-purpose thermal management techniques applicable to all processor paradigms we propose to take advantage of SMT's unique flexibility of having multiple threads. By selectively managing the execution of available threads we see an opportunity to adaptively counteract and prevent hot spots. Our work uses the Turandot simulator to model an SMT-supporting POWER5^{TM}-like processor and the HotSpot 2.0 tool to simulate thermal behavior. With it, we examine the performance of our SMT-specific adaptive thread control mechanisms as compared to conventional dynamic thermal management techniques. We find that when multiple heterogeneous programs are available in the workload, thermal-aware issue policies provide a significant power-performance benefit; they average 44% ED^2 reduction when aggressively operating near the thermally limited region. We observe the inherent tradeoffs between such performance advantages and thread fairness, and test this design as an instruction fetch policy as well as an adaptive register renaming technique.

1 Introduction

As transistor densities continue to increase in modern processors, on-chip temperature management quickly emerges as a performance-constraining bottleneck. This has spawned the necessity for *temperature-aware design* in addition to conventional performance-oriented and power-aware design [24]. A number of adaptive control methods have been proposed for temperature management in uniprocessors. These include global management techniques such as dynamic voltage and frequency scaling (DVFS) and global clock gating, as well as more localized techniques such as fetch/dispatch throttling and register-file throttling [2, 8, 23]. While such techniques have been shown to greatly aid thermal management, recurring challenges involve optimizing the necessary power/performance tradeoffs, ensuring sustained performance, and particularly dealing with hot spots—small sections of a chip attaining temperatures significantly higher than the chip's overall temperature.

For examining thermal issues it is important to explore the problem in the context of prominent architectural paradigms, thus we explore this issue in simultaneous-multithreaded (SMT) processors. SMT cores seek greater performance by densely packing issue slots and hence can be cause for thermal stress. Our work explores the idea of taking advantage of SMT's added flexibility due to the availability of multiple threads. As a localized technique, we propose that selectively fetching among different programs can allow thermal hot spots to be better controlled and prevented. In our experiment we show that adaptive thread management can tightly control temperature, which has implications for better thermal management and overall reliability [26]. Also, as a localized microarchitectural mechanism, application and design of such adaptive control can work independently or in conjunction with *global* thermal management techniques such as DVFS.

Our specific contributions are as follows:

- We characterize several benchmarks based on their respective hot spot behaviors. We find that for our processor configuration, each program's hot spot behavior can be characterized largely by its integer register file intensity and floating point register file intensity.
- We propose and evaluate an online adaptive fetch algorithm to take advantage of these heterogeneous characteristics when threads are mixed through SMT. We find that when operating in the thermally limited region, our algorithm reduces the occurrence of thermal emergencies resulting in increased performance by an average of 30% and

ED^2 product reduction on the order of 40%. Furthermore, this is a *local* temperature management policy which targets hot spots and can be used in combination, rather than in competition, with global thermal management such as DVFS.

- We repeat these experiments with a similar adaptive algorithm based on selective register naming instead of instruction fetching. For this alternate mechanism, which operates at a later stage in the pipeline, we find correlated but comparatively smaller performance improvements: roughly 70% as effective.

The remainder of this paper is structured as follows. Section 2 discusses related work and our motivation to extend upon these studies. Section 3 presents our simulation infrastructure and methodology. In Section 4 we explain our adaptive fetch policy and show our experimental results in terms of measured performance effects and energy savings. In Section 5 we perform similar experiments from an adaptive register renaming perspective and compare to the corresponding fetch throttling or adaptive fetch results. In Section 6 we conclude and discuss directions for future work.

2 Background and Related Work

Simultaneous multithreading is an architectural paradigm that involves issuing instructions such that multiple threads on a single core closely share resources [29]. Various implementations of SMT are now available in several commercial processors [3, 12, 27].

A number of works have examined the power and energy properties of SMT without regard to spatial temperature analysis [13, 16, 20, 21]. Interestingly these studies tend to explore SMT in comparison to *chip multiprocessing* (CMP), a different architectural paradigm that is attractive due to also achieving multithreaded behavior. Several other works extend beyond these and examine the thermal properties of SMT [5, 15, 19]. In addition to characterizing SMT's thermal behavior, a number of thermal management techniques for SMT processors have been proposed and studied. For instance, Li et al. [15] experiment with dynamic voltage scaling and localized throttling techniques. However, all their tested techniques are applicable to superscalar processors and other paradigms as well; hence, they do not explore SMT-specific constructions. Powell et al. [19] explore SMT thermal management in the context of hybrid SMT-CMP systems and they propose scheduling schemes for optimal scheduling on thermally constrained designs. However, their design intervenes only through the operating system and they do not explore more fine-grain techniques that could enable thermal management without requiring context switches.

Albonesi et al. propose SMT-specific extensions targetting another reality of physics for modern processors—the inductive noise problem [6]. Similar to our reasoning, they see SMT providing an opportunity to exploit program diversity in order to counteract with adaptive control.

Hasan et al. [9] propose a mechanism that strongly relates to the design in this paper. They envision a scenario whereby a malicious thread may cause a microarchitectural Denial of Service (DoS) attack, and propose remedies for detecting and mitigating the effects of such attacks. However, they do not examine how to optimize SMT operation in terms of naturally occurring thermal stress. Our work here explores this as a general problem to be addressed as processor designs are bound to become more thermally stressed in the future and operate under thermally constrained conditions. Our proposed framework manages to generalize protective thermal arbitration to all programs, *including* programs that could potentially be intended for malicious attacks.

3 Methodology

3.1 Simulation Framework

We model a detailed out-of-order CPU resembling a single-core portion of the IBM POWER4$^{\text{TM}}$processor with SMT support as is used in the POWER5$^{\text{TM}}$. Our simulation framework is based on the IBM Turandot simulator as part of the Microarchitectural Exploration Toolset (MET) [17]. Dynamic power calculations are provided by PowerTimer, an add-on for Turandot that provides detailed power measurements based on macroblock formations derived from low-level RTL power simulations [1]. Both Turandot and PowerTimer have been extended for SMT support as detailed in [16]. Integrated with this is the HotSpot 2.0 [11, 24] temperature modeling tool to provide spatial thermal analysis.

We model a single-core processor with SMT support on 0.18μ technology. This design level is known to already create significant hot spot effects, a problem which becomes even more prominent at smaller feature sizes. Our design parameters are shown in Table 1. Although PowerTimer is directly parameterized based on these options, HotSpot naturally requires additional input to describe the processor's spatial layout. This floorplan is shown in Figure 1.

Since PowerTimer does not model leakage current by default, an added modification is to model leakage through the area-based empirical equation in [10]. Thus the leakage power of each structure is calculated only by its area and time-dependent temperature. Although more diverse and accurate leakage models do exist, this equation is sufficient to model the temperature dependence and quickly derive leakage estimates for all processor structures.

3.2 Benchmarks

We analyze workloads based on ten benchmarks obtained from the SPEC 2000 benchmark suite. We have

Global Design Parameters	
Process Technology	0.18μ
Supply Voltage	1.2 V
Clock Rate	1.4 GHz
Organization	single-core
Core Configuration	
SMT Support	2 threads
Dispatch Rate	5 instructions per cycle
Reservation Stations	mem/int queue (2x20), fp queue (2x5)
Functional Units	2 FXU, 2 FPU, 2 LSU, 1 BRU
Physical Registers	120 GPR, 90 FPR
Branch Predictor	16K-entry bimodal, 16K-entry gshare, 16K-entry selector
Memory Hierarchy	
L1 Dcache	32 KB, 2-way, 128 byte blocks, 1-cycle latency
L1 Icache	64 KB, 2-way, 128 byte blocks, 1-cycle latency
L2 I/Dcache	2 MB, 4-way LRU, 128 byte blocks, 9-cycle latency
Main Memory	77-cycle latency

Table 1. Design parameters for modeled CPU.

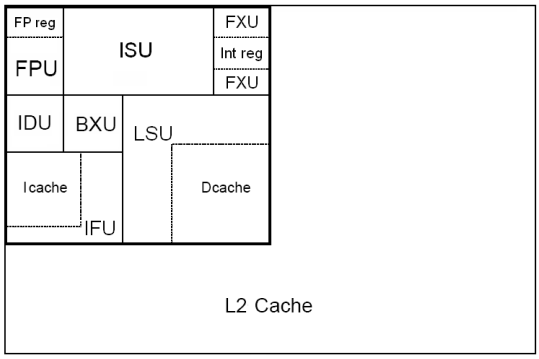


Figure 1. Floorplan input to HotSpot 2.0, as also used in [15].

chosen five programs from the integer-based SPECint portion and the other five are from SPECfp, as depicted in Table 2.

For outcomes of mixing different programs through simultaneous multithreading it has been shown that the end performance effects can be predicted somewhat based on characteristics of the individual applications [15, 28, 25]. Thus we also characterize our individual test programs before deciding upon which combinations to mix through multithreading. While hot spots can be unmanageable if their locations vary unpredictably with time, various simulation results [5, 7, 15, 24] have indicated that for particular processor designs hot spots predictably tend to occur in a handful of locations.

In our design, we find that almost universally the hottest portion of the chip is either the fixed-point execution (FXU) register file or the floating point (FPU) register file. Thus each of the benchmarks are measured in terms of their thermal intensity for these two chip locations. This measurement is done in advance by executing the programs *without* thermal control and examining the steady-state and final temperatures on these units. Programs that showed steady-state temperatures above 93°C on units have been marked as such in the two rightmost columns of Table 2. For our dynamic policy, later described in Section 4.1, it shall be necessary to know the heating characteristics of the running programs. For this, we observe a direct correlation between each program's register file heating characteristics and the number of register file accesses recorded, and we are able to universally use this observed ratio in our dynamic policy when applied to any workload.

We then use this data in deciding how to appropriately create a set of ten SMT-based workloads. First, we would like to mix programs which show opposite thermal behaviors since these give the greatest potential for adaptive thermal control. These include mixing integer intensive programs with floating-point intensive programs. For the other end of the spectrum, we also include several test cases which lack thermal heterogeneity, such as pairs of floating point benchmarks and pairs or integer-only benchmarks. In such scenarios we might not expect a significant benefit from thread-sensitive thermal control, but it is important to show that our algorithm can at least be ensured not to be detrimental in these cases. A list of these chosen workloads and their corresponding qualitative characterizations can be found in Table 3.

In order to simulate only representative portions of these programs, we use SimPoint [18, 22] with sampling intervals of 100 million instructions in order to obtain all relevant traces executed in our experiments. To simulate relevant temperature behavior on such a short time interval, we choose an operating point and thermal threshold such that thermal triggers come into play approximately 60% of the time for most our test workloads. This can also be described as a "duty cycle" of 40% [19]. The time spent in thermal emergency mode shall naturally decrease when applying our adaptive policies.

3.3 Metrics

As one measure of the performance impact of our technique, we use the criterion of *weighted speedup* as described by Snavely and Tullsen [25] shown below.

$$Weighted\ Speedup = \sum \frac{IPC_{SMT}[i]}{IPC_{normal}[i]}$$

This is intended to be a fair comparison between two executions and prevents biasing the metric on policies that execute unusual portions of high-ILP or low-ILP threads. Note that the $IPC_{SMT}[i]$ is only a portion of the multithreaded system's total IPC.

In these experiments the $IPC_{normal}[i]$ denominator is measured under thermally limited conditions. To be specific, all executions start with temperature profiles where both register files are just barely below the thermal threshold of 85°C. Under these conditions we

name	benchmark suite	function	FXU-reg intensive	FPU-reg intensive
188.ammp	SPECfp	computational chemistry	N	Y
173.applu	SPECfp	computational fluid dynamics/physics	N	Y
191.fma3d	SPECfp	mechanical response simulation	Y	Y
178.galgel	SPECfp	computational fluid dynamics	Y	Y
176.gcc	SPECint	C language compiler	Y	N
164.gzip	SPECint	compression	N	N
181.mcf	SPECint	mass transportation scheduling	Y	N
177.mesa	SPECfp	3-D graphics library	Y	N
197.parser	SPECint	word processing	N	N
300.twolf	SPECint	lithography placement and routing	Y	N

Table 2. SPEC 2000 benchmarks as selected for this experiment, listed alphabetically.

workload	thermal heterogeneity	reason
ammp-gzip	significant	floating point benchmark mixed with an integer benchmark.
ammp-mcf	significant	floating point benchmark mixed with an integer benchmark.
applu-parser	moderate	can exploit parser's extremely low IPC to cool either hot spot.
applu-twolf	significant	floating point benchmark mixed with an integer benchmark.
fma3d-galgel	small	both benchmarks are high-intensity on both register files.
fma3d-twolf	small	both benchmarks are integer-intensive.
galgel-mesa	moderate	both benchmarks are integer-intensive, but mesa is greater.
gcc-mesa	small	two integer benchmarks.
gcc-parser	moderate	two integer benchmarks, but parser's slowness needs management.
gzip-mcf	small	two integer benchmarks.

Table 3. Multithreaded benchmark mixes. Pairs with a higher degree of thermal heterogeneity show greater promise in benefitting from SMT-specific adaptive thermal management.

find that a number of workloads go into thermal arrest about 60% of the time, and this affects the denominator in the above equation. This method is largely different from other works such as [5, 25, 28] where weighted speedup is measured assuming the baseline single-threaded executions are not thermally constrained in any way. We believe that for the purpose of this study, however, our baseline is more appropriate since we seek to analyze behavior particularly in the thermally limited region. Weighted speedup is meant to qualify as a fair raw performance metric similar to how IPC is sometimes used in uniprocessor comparisons. While the weighted metric is arguably more qualified for our purposes, in most of our results the overall workload IPC is strongly correlated to weighted speedup anyway. However, it is clear that weighted speedup can dramatically increase despite thread commitment policies being on the whole "unfair". Thus, we also directly present the ratios of thread retirement on a per-thread basis for each of our tested workloads.

In order to appropriately measure performance from a power-aware perspective, for our second main metric we use the established energy·delay-squared product (ED^2). This now widely used metric realistically takes into account tradeoffs between power and energy in the context of DVFS, where scaling the voltage can have a cubic effect on power reduction. Since our proposed adaptive policy is a local mechanism, it can still be combined with global power reduction techniques such as DVFS, thus making this a relevant evaluation metric. Since we are measuring workloads that complete with different instruction counts and instruction mix ratios on different parameterizations, we must normalize the ED^2 metric to a *per instruction* basis. We use the following formula to calculate this metric from the IPC and energy per instruction, EPI.

$$ED^2 = \frac{EPI}{IPC^2 * clock\ frequency^2}$$

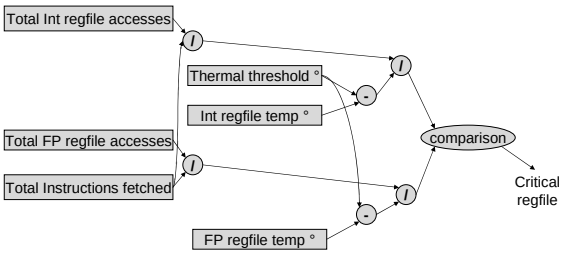
4 Adaptive Thermal Control

4.1 Adaptive Control Algorithm Overview

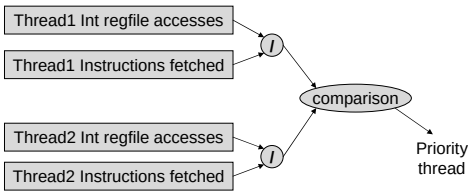
Our adaptive control is based on the input of temperature sensors that exist in many modern commercial processors. Although the exact placement of the POWER5^{TM}processor's 24 available sensors is unknown [3], it is reasonable to assume that at least two of these would be allocated to the register file locations which are primary potential hot spots. We use 85° C as the threshold temperature for enacting thermal control. As modern commercial microprocessors tend to list maximum allowable operating temperatures in the range of 70 to 90° C [4] we feel this is a reasonable choice. Utilizing the dynamically profiled thread behavior information, our decision algorithm for adaptive thread selection is implemented on top of this as follows:

For the actual adaptive technique of dynamic thermal management, we modify the default round-robin SMT fetch policy originally implemented for Turandot in [16]. Our modifications target thermal control logically by avoiding integer-intensive benchmarks when the FXU register file's temperature appears more likely to reach the temperature threshold, and likewise to reduce the execution rate of floating point intensive benchmarks when the FPU register file goes

above its threshold. In order for the processor to identify whether running programs are integer-intensive or floating point intensive, we must dynamically sample hardware event counters. As mentioned in Section 3.2, we are able to exploit a direct correlation between register file accesses and the long-term steady state register file temperature. In [19], Powell et al. also use counter information as such to predict heating behavior for key resources [19], and recent work by Lee and Skadron has shown that hardware performance counters can be reliably used to predict temperature effects on real systems [14].



(a) Portion of our algorithm that determines which unit is in thermal danger.



(b) Decision algorithm for which thread is selected if the integer (FXU) register file is judged to be in danger. The decision process for the floating point register file is identical.

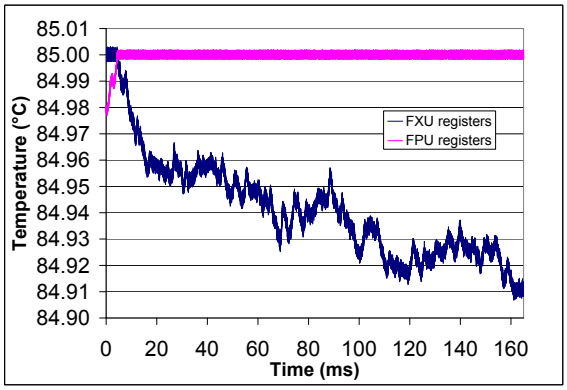
Figure 2. Block diagrams demonstrating the calculation and decisions in our algorithm. These also reflect the added components used in a hardware design.

When the processor is not in thermal arrest mode, the difference between the thermal threshold and the integer register file's temperature is calculated. At the same time, from profiling we obtain the average number of integer register file accesses per fetched instruction for each of the two threads. Using a calibrated threshold—in terms of PowerTimer's internal access counters—we decide whether the integer register file is in danger of approaching our specified maximum temperature (85°). We also do all of the above for the floating point register file and compare to see which unit is potentially in danger. The steps necessary to calculate and decide this are depicted graphically in Figure 2 (a). Our adaptive policy then takes effect. Its goal is to choose instructions from the thread that is either likely to cool or less quickly heat the hotter of the two register files. Once the potentially hotter of the two units is identified, the decision as to which thread to pick from is decided by choosing the thread measured to be less intensive on the integer register file—or floating point register file, if applicable—based on the threads' dynamically profiled measure of register file accesses per issued instruction. This second stage of the decision process is depicted in Figure 2 (b). These calculations are done only once per temperature measurement cycle, and are precalculated with delay in such a way that it does not affect the fetch logic's critical path.

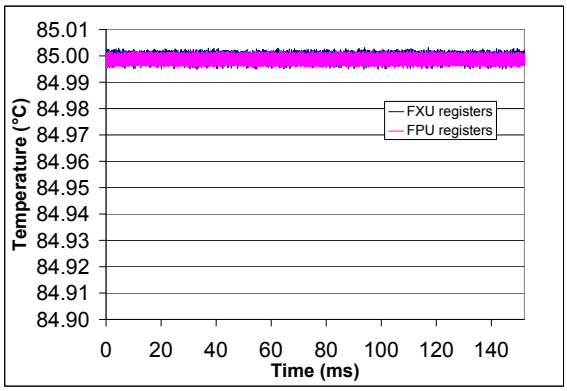
Fetch priority adjustment is in many ways an extension of basic fetch throttling on a uniprocessor. Also known as toggling, throttling involves simply disabling instruction fetch whenever a section of the processor surpasses the specified thermal threshold [2]. Once this mechanism has been triggered, ideally the processor would quickly cool down until it goes below the thermal threshold and can continue normal operation. Fetch throttling thus forms the comparison baseline for our measurements. In actuality, our fetch priority adjustment system is not an alternative but rather runs in combination with fetch throttling. Since thermal stability cannot be ensured if instructions are always issued—as is the case when all available threads are thermally intensive—it is necessary for our design to have a backup policy to fall upon in order to guarantee prevention of thermal violations. Figure 3 shows a sample of the effects of thermal management under our proposed algorithm from a time-dependent perspective. In the baseline fetch throttling example of 3 (a) one hot spot can remain the primary performance hindrance, while with our adaptive algorithm in 3 (b) instructions from each thread can be issued such that the two key hot spot temperatures remain close.

4.2 Adaptive Control Algorithm: Other Issues and Discussion

To avoid unpredictable cases of thread starvation, we allocate a portion of cycles where the default fetch policy holds regardless. For our policy labeled "moderate", the first two cycles out of every four cycles default to the standard alternation among threads (round-robin) policy. This ensures a degree of thread fairness fairly close to the original policy, but at the potential expense of poorer thermal management. Our "aggressive" policy allocates only the first two out of every 16 cycles for defaulting to the round-robin policy, pushing a stronger tradeoff between thread fairness and thermal management. While our current fallback policy is round-robin, for future work we hope to extend our framework to use more real world-applicable fetch policies including ICOUNT [29]. Such designs, which were originally aimed for aggressive performance, may become more severely penalized under thermally limited conditions and hence would likely benefit more from our temperature-aware policies.



(a) Baseline fetch throttling thermal control.



(b) Temperature-aware thread fetch policy.

Figure 3. Transient hot spot temperatures for fma3d-twolf workload under our baseline and adaptive policy. Swings as depicted in (a) are extrapolated to longer time intervals amount to possible temperature changes on the order of 5° every 10 seconds.

For identifying the heat behavior of each thread, we must sample its execution through performance counters at runtime. Our current dynamic profiling utilizes event counts ranging 100 temperature measurement cycles (1,000,000 CPU cycles) earlier up until the point of the most recent temperature sample. Being two orders of magnitude larger than the temperature measurement cycle, we ensure that profiling functions properly only as uninterfered background information for the decision algorithm. However, the profiling data does not extend too far back into past execution, because earlier program behavior can likely be unrepresentative of future behavior.

We do not model sensor error, although sensor delay is modeled as temperature is recalculated only every 10,000 cycles. At the given clock rate this amounts to about 6 μs. Thus any hardware necessary for recalculating the temperature and feeding it to the control logic cannot be expected to affect the critical path of the pipeline, as the result is precalculated and fed in with appropriate delay. We find under this model that it usually takes between one and three measurement cycles (10,000 to 30,000 CPU cycles) to fall back below the thermal threshold after each thermal threshold breach is detected. Despite thermal emergencies occurring and being dealt often throughout execution, there is no additional delay penalty for enacting thermal control. Compared to DVFS, this is a key advantage of pure microarchitectural techniques as highlighted by [2].

Heo et al. [10] have shown that designs enacting thermal control on a sufficiently fine-grain interval pose an advantage for tightly controlling temperatures although they can be more costly in terms of other design factors. However, it seems feasible that this mechanism could be moved to the operating system level, as Powell et al. have demonstrated that thermal fluctuations happen on a sufficiently coarse grain time interval adequate to be managed by the OS [19]. Hybrid techniques involving both the microarchitecture and OS are also a possible implementation. To be specific, prioritized fetching and renaming can be performed by the microarchitecture, while numerical specifications of those process priorities can be dictated by the OS depending on thermal conditions. While we focus on a purely rapid-response microarchitectural mechanism in this paper, the necessary granularity of operation remains an open question for future study.

For implementing the control algorithm in real hardware, event counters are necessary to measure (in total as well as on a per-thread basis) integer register file accesses per cycle, floating point accesses per cycle, and number of instructions fetched per cycle. Secondly, calculation hardware is needed including adders, a division unit, and necessary decision logic. Note that although our algorithm as depicted in Figure 2 shows as many as eight dividers, in reality only a single shared divider is necessary since speed of calculation is not critical. Since these calculations would be invoked only once for every temperature measurement cycle, the energy overhead is negligible. For perspective, modern DVFS solutions employ PID-based hardware which involves even more additional gates but also has insignificant energy overhead while not affecting the microprocessor pipeline's critical path.

4.3 Results and Observations

In this section we examine the benefits of temperature-aware adaptive thread priority management. Table 4 lists performance and power metrics for all mixes under the baseline control method. The weighted speedup for each of these mixes is small, notably less than 1.0 in all cases. This signifies a *cost* associated with simultaneous multithreading, and it is primarily due to operating in the thermally limited region. It is this cost we seek to address. Although all mixes have weighted speedups greater than 1.0 when operating below the thermally limited region,

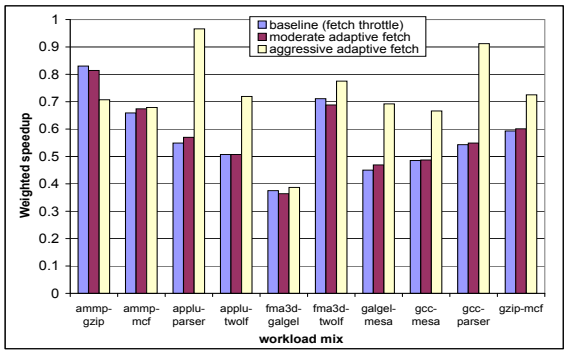
the prospect of running into thermal control due to more issued instructions makes SMT actually detrimental to performance in this region. For these executions the bottleneck hot spots are still the integer and floating point register files where one or the other hovers at the thermal threshold of 85°C. The overall chip temperature as reflected by its large L2 cache remains at approximately 52°C, more than 30° less.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)
ammp-gzip	0.820	57.8%/42.2%	0.830	2.83e-26
ammp-mcf	0.410	66.7%/33.3%	0.659	2.23e-25
applu-parser	0.527	63.5%/36.5%	0.549	9.87e-26
applu-twolf	0.486	60.6%/39.4%	0.507	1.27e-25
fma3d-galgel	0.583	43.4%/56.6%	0.375	7.48e-26
fma3d-twolf	0.716	60.4%/39.6%	0.711	4.17e-26
galgel-mesa	0.708	60.8%/39.2%	0.450	4.27e-26
gcc-mesa	0.494	53.0%/47.0%	0.485	1.22e-25
gcc-parser	0.485	62.0%/38.0%	0.543	1.30e-25
gzip-mcf	0.304	57.1%/42.9%	0.593	5.25e-25

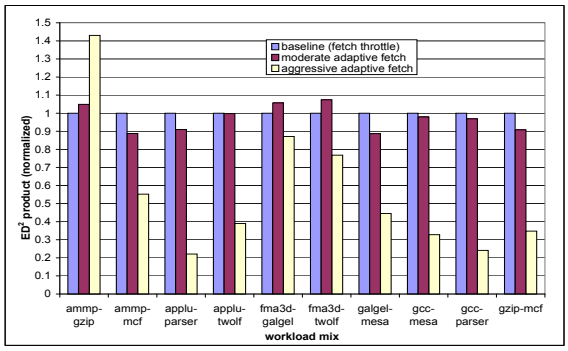
Table 4. Baseline results for fetch toggling based DTM without adaptive thread control.

Improving on this baseline, Table 5 lists performance and power metrics for all mixes under adaptive thread fetching for our moderate and aggressive-level policies. Figure 4 pulls together the primary parameters as presented in Tables 4 and 5 and presents these results graphically. Note that in most cases where heterogeneously behaved programs are mixed, we see a 30-40% IPC improvement with a similar increase in weighted speedup. This performance improvement is directly caused by a corresponding reduction in the number of thermal emergencies. The ED^2 reduction is related to this parabolically and can be exlained as follows. Dynamic power increases proportionally with higher IPC, but this does not significantly reduce EPI since the amount of work performed per instruction remains the same. Leakage power, on the other hand, remains mostly unchanged since our overall chip temperature remains largely unaffected, resulting in somewhat lower energy *per instruction* as leakage in this model constitutes only about 25% of total power. Thus the key factor causing a parabolically correlated decrease in ED^2 reduction is the delay term squared.

As expected, we find that our adaptive fetch technique offers the biggest improvement in cases allowing a high degree of thermal variety in workload mixes. For other cases such as gzip-mcf and gcc-mesa (integer only), we see there is actually a significant performance potential despite the constituent programs being similar in terms of register file usage. The exploitable difference here is perhaps that although neither program uses floating point operations, these programs already possess much imbalance in terms of their frequency of integer accesses. One workload, ammp-gzip, shows a decrease in performance under our algorithm. Although this at first seems surprising since it is a heterogeneous workload—containing an integer benchmark and one floating-point benchmark—that should



(a) Weighted speedup for all workloads under three thermal-aware fetch policies.



(b) Corresponding normalized ED^2 product for these workloads.

Figure 4. Weighted speedup and ED^2 for fetch-based dynamic thermal management.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)
ammp-gzip	0.806	58.9%/41.1%	0.814	2.97e-26
ammp-mcf	0.427	68.1%/31.9%	0.674	1.98e-25
applu-parser	0.545	61.8%/38.2%	0.570	8.99e-26
applu-twolf	0.486	61.0%/39.0%	0.507	1.27e-25
fma3d-galgel	0.572	41.4%/58.6%	0.364	7.90e-26
fma3d-twolf	0.698	62.6%/37.4%	0.688	4.47e-26
galgel-mesa	0.738	60.7%/39.3%	0.469	3.79e-26
gcc-mesa	0.498	50.5%/49.5%	0.487	1.19e-25
gcc-parser	0.490	60.4%/39.6%	0.549	1.25e-25
gzip-mcf	0.314	59.0%/41.0%	0.601	4.77e-25

(a) Moderate adaptive fetch management.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)
ammp-gzip	0.720	70.3%/29.7%	0.707	4.05e-26
ammp-mcf	0.498	78.4%/21.6%	0.679	1.23e-25
applu-parser	0.892	47.5%/52.5%	0.966	2.19e-26
applu-twolf	0.675	51.1%/48.9%	0.719	4.96e-26
fma3d-galgel	0.611	40.8%/59.2%	0.387	6.52e-26
fma3d-twolf	0.784	62.1%/37.9%	0.775	3.20e-26
galgel-mesa	0.936	35.6%/64.4%	0.692	1.90e-26
gcc-mesa	0.719	22.5%/77.5%	0.666	3.99e-26
gcc-parser	0.787	32.8%/67.2%	0.912	3.12e-26
gzip-mcf	0.436	72.3%/27.7%	0.725	1.83e-25

(b) Aggressive adaptive thread management.

Table 5. Complete data for workload behavior under our adaptive thread fetching policy.

have potential for balancing, upon inspection the cause is that the baseline case using throttling happens to be already very balanced with starting and ending temperatures for each register file remaining close to each other. This most likely happens by chance; a larger or different program trace for the programs are selected the temperatures could easily imbalance without adaptive thread management.

The potential cost of our adaptive policy is reduced thread execution fairness as compared to the basic round-robin policy. Overall, we find that the moderate adaptive policy performs better than the baseline with an average of only 1% improvement in terms of weighted speedup or IPC. Our aggressive policy performs significantly better than the moderate policy showing an average of 30% improvement in terms of weighted speedup. The ED^2 product, strongly correlated, averages 44% reduction under the aggressive adaptive policy.

5 Adaptive Register Renaming

5.1 Design Description

Our second set of experiments is much like the first, except it involves adaptive control at a later stage of the pipeline, namely the register renaming logic. Our adaptive rename policy is exactly the same as explained earlier for adaptive fetch control, except instead of being fetch-based it controls the priority at which a thread receives the register renaming service. For deciding which thread to give renaming priority to on each cycle, we use the same decision policy as depicted in Figure 2. When the decision to rename registers for only a particular thread is decided on any given cycle, the register renamer hardware maps registers only for the selected thread, effectively stalling services for the other thread. Likewise, instead of fetch throttling serving as our baseline thermal control method, we compare against basic rename throttling [15] instead. This involves simply disabling the rename logic when the processor appears above its thermal threshold. A difference, and possible benefit from this technique, is that it operates closer to the hot spot of interest, namely the register file. A clear drawback is that throttling at a later stage of the pipeline allows instructions to enter the pipeline and consume resources.

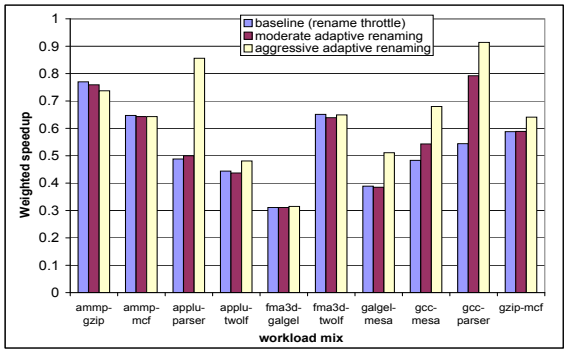
5.2 Results and Observations

Our baseline results regarding rename throttling without adaptive register renaming are shown in Table 6. We find the efficacy of this alternative thermal management technique to be on the same order of efficacy as fetch throttling, a result consistent with [15].

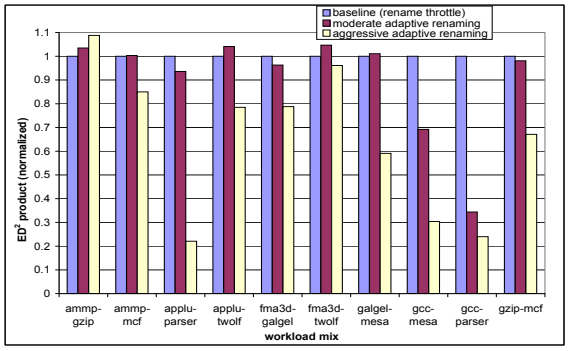
We enact the adaptive register renaming strategy described in 5.1. As with our other fetch-based experiments, note that this is not an alternative to basic register rename throttling but rather is operating on top of the parent policy so as to ensure thermal stability. Table 7 shows all corresponding data for the adaptive renaming experiments, and likewise for comparison Figure 5 brings together the main results of Tables 6 and 7 to compare graphically. The pattern of measurable performance improvement in terms of ED^2 is much the same as is found from our fetch-based experiments. That is, we see roughly the same pattern of

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)
ammp-gzip	0.760	57.7%/42.3%	0.770	3.56e-26
ammp-mcf	0.402	66.6%/33.4%	0.647	2.41e-25
applu-parser	0.467	63.2%/36.8%	0.488	1.44e-25
applu-twolf	0.425	60.3%/39.7%	0.444	1.92e-25
fma3d-galgel	0.484	43.1%/56.9%	0.311	1.32e-25
fma3d-twolf	0.655	60.3%/39.7%	0.651	5.46e-26
galgel-mesa	0.612	60.8%/39.2%	0.389	6.64e-26
gcc-mesa	0.492	53.0%/47.0%	0.483	1.28e-25
gcc-parser	0.486	61.8%/38.2%	0.544	1.32e-25
gzip-mcf	0.302	57.1%/42.9%	0.588	5.49e-25

Table 6. Baseline results for rename-throttling based DTM without adaptive thread-specific renaming.



(a) Weighted speedup for all workloads under the three thermal-aware renaming policies.



(b) Corresponding ED^2 product for these workloads, normalized.

Figure 5. Weighted speedup and ED^2 for register renaming-based dynamic thermal management.

performance gains in certain workloads. As mentioned earlier, a drawback expected from throttling at the rename stage is that the register renamer is a later stage of the pipeline, thus unlike fetch management it gives more potential for unwanted instructions to enter the pipeline and consume resources while throttled. Despite this possible downside, the potential for thermal control at this pipeline stage in addition to the fetch stage appears quite viable.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)
ammp-gzip	0.751	58.4%/41.6%	0.759	3.69e-26
ammp-mcf	0.401	67.0%/33.0%	0.643	2.42e-25
applu-parser	0.477	62.7%/37.3%	0.500	1.35e-25
applu-twolf	0.419	61.3%/38.7%	0.437	2.00e-25
fma3d-galgel	0.491	40.5%/59.5%	0.311	1.27e-25
fma3d-twolf	0.645	61.3%/38.7%	0.639	5.71e-26
galgel-mesa	0.610	61.8%/38.2%	0.385	6.71e-26
gcc-mesa	0.558	47.7%/52.3%	0.543	8.83e-26
gcc-parser	0.703	56.8%/43.2%	0.792	4.55e-26
gzip-mcf	0.304	57.5%/42.5%	0.589	5.38e-25

(a) Moderate adaptive register renaming.

mix	IPC	thread retire ratio	weighted speedup	ED^2 ($\frac{J \cdot s^2}{instr^3}$)
ammp-gzip	0.736	62.5%/37.5%	0.737	3.88e-26
ammp-mcf	0.423	70.9%/29.1%	0.643	2.05e-25
applu-parser	0.786	46.1%/53.9%	0.856	3.19e-26
applu-twolf	0.461	61.7%/38.3%	0.481	1.51e-25
fma3d-galgel	0.526	31.9%/68.1%	0.315	1.04e-25
fma3d-twolf	0.663	64.9%/35.1%	0.649	5.24e-26
galgel-mesa	0.733	45.9%/54.1%	0.511	3.92e-26
gcc-mesa	0.733	23.4%/76.6%	0.680	3.88e-26
gcc-parser	0.788	32.7%/67.3%	0.914	3.18e-26
gzip-mcf	0.346	62.1%/37.9%	0.641	3.68e-25

(b) Aggressive adaptive register renaming.

Table 7. Complete data for workload behavior under our adaptive register renaming policy.

6 Conclusions and Future Work

This study proposes and tests a novel form of adaptive DTM specific to SMT processors. We have shown that adaptive thread fetching can predictably control temperature of hot spots at a fine grain level. We have found thread priority management providing a weighted speedup performance increase over our conventional fetch toggling technique by an average of 30%, and ED^2 reductions averaging 44% for our test cases. Our analogous experiments dealing with adaptive renaming found strikingly similar results averaging 23% weighted speedup improvement and 35% ED^2 reduction.

Our work demonstrates a heuristic algorithm for a simple case of two primary hot spots on an SMT processor. Future process technologies bring greater thermal challenges including wider gaps between overall chip temperature and localized hotspots, we expect this to worsen and create increased demand for smart thermal control applicable to varied workloads. Such systems pose a challenge but a wider variety of hot spots also brings potential for more advanced adaptive control methods.

Our proposed algorithm makes a clear tradeoff between baseline thread fairness and sustaining performance. It is most applicable in systems which allow a wide degree of thread priority and scheduling freedom. This would include systems such as scientific computing environments where many huge workloads are queued up without strict process priorities. One can also envision, for example, a thermally constrained server system where one might find it more appropriate to fairly allocate user time based on its thermal cost (power) rather than direct CPU-cycle cost. A mechanism such as this one directly enables such an energy-guided quota. A general-purpose policy such as this could obviate overly specific protection against malicious thermal attacks such as described in [9].

For our future work we wish to explore these adaptive techniques in the context of relevant processor paradigms. Since SMT is now commonly coupled in CMP systems—and such hybrid systems are supported by this Turandot simulator—we wish to extend upon the work here to test adaptive control in such complex systems. Furthermore, our current construction is limited to 2-context SMT and does not readily scale to greater numbers of threads. While the logic for comparing two threads based on a critical resource's temperature extended to sort multiple threads, it is then not clear how to partition multiple threads practically in terms of allowed execution share. Other possibilities for extending this work to test it in relevant contexts involve combining with complex fetch policies such as ICOUNT, and combinining these localized DTM techniques with global mechanisms such as DVFS. Furthermore, the algorithm presented here is entirely heuristic by nature, and without formal analysis this prevents us from knowing the full potential. We hope to apply control theory to better explore ideas for hot spot management from an analytical framework.

7 Acknowledgements

We are grateful to Yingmin Li for providing source code modifications to integrate HotSpot with the Turandot simulator. We would also like to thank the anonymous reviewers for their helpful comments. This work is supported in part by grants from NSF, Intel, and SRC.

References

[1] D. Brooks, P. Bose, S. Schuster, H. Jacobson, P. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. Cook. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

[2] D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In *HPCA '01: Proceedings of the Seventh International*

Symposium on High-Performance Computer Architecture, page 171, 2001.

[3] J. Clabes, J. Friedrich, M. Sweet, J. Dilullo, S. Chu, D. Plass, J. Dawson, P. Muench, L. Powell, M. Floyd, B. Sinharoy, M. Lee, M. Goulet, J. Wagoner, N. Schwartz, S. Runyon, G. Gorman, P. Restle, R. Kalla, J. McGill, and S. Dodson. Design and Implementation of the POWER5^{TM}Microprocessor.

[4] CPU Maximum Operating Temperatures. `http://www.gen-x-pc.com/cputemps.htm`. Gen-X PC, 2005.

[5] J. Donald and M. Martonosi. Temperature-Aware Design Issues for SMT and CMP Architectures. In *WCED-5: Proceedings of the 5th Workshop on Complexity-Effective Design*, June 2004.

[6] W. El-Essawy and D. H. Albonesi. Mitigating Inductive Noise in SMT Processors. In *ISLPED '04: Proceedings of the Proceedings of the 2004 International Symposium on Low Power Electronics and Design (ISLPED'04)*, pages 332–337. IEEE Computer Society, 2004.

[7] S. Ghiasi and D. Grunwald. Design Choices for Thermal Control in Dual-Core Processors. In *WCED-5: Proceedings of the 5th Workshop on Complexity-Effective Design*, June 2004.

[8] S. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the Impact of Increasing Microprocessor Power Consumption. *Intel Technology Journal*, Q1, 2001.

[9] J. Hasan, A. Jalote, T. N. Vijaykumar, and C. Brodley. Heat Stroke: Power-Density-Based Denial of Service in SMT. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 166–177. IEEE Computer Society, 2005.

[10] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2003.

[11] W. Huang, M. R. Stan, K. Skadron, K. Sankaranarayanan, S. Ghosh, and S. Velusamy. Compact Thermal Modeling for Temperature-Aware Design. In *DAC: Proceedings of 41st Design Automation Conference (DAC)*, pages 878–883, June 2004.

[12] Hyper-Threading Technology. `http://www.intel.com/technology/hyperthread/`. Intel Corporation, 2005.

[13] S. Kaxiras, G. Narlikar, A. D. Berenbaum, and Z. Hu. Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads. In *CASES '01: Proceedings of the 2001 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*, pages 211–220. ACM Press, 2001.

[14] K.-J. Lee and K. Skadron. Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors. In *Workshop on High-Performance, Power-Aware Computing (HP-PAC)*, Apr. 2005.

[15] Y. Li, D. Brooks, Z. Hu, and K. Skadron. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, Feb. 2005.

[16] Y. Li, D. Brooks, Z. Hu, K. Skadron, and P. Bose. Understanding the Energy Efficiency of Simultaneous Multithreading. In *ISLPED '04: Proceedings of the 31st Annual International Symposium on Low Power Electronics and Design*, pages 44–49. ACM Press, 2004.

[17] M. Moudgill, J.-D. Wellman, and J. H. Moreno. Environment for PowerPC Microarchitecture Exploration. *IEEE Micro*, 19(3):15–25, May/June 1999.

[18] E. Perelman, G. Hamerly, and B. Calder. Picking statistically valid and early simulation points. In *PACT '03: Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, page 244. IEEE Computer Society, 2003.

[19] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run: Leveraging SMT and CMP to Manage Power Density Through the Operating System. In *ASPLOS-XI: Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 260–270. ACM Press, 2004.

[20] R. Sasanka, S. V. Adve, Y.-K. Chen, and E. Debes. The Energy Efficiency of CMP vs. SMT for Multimedia Workloads. In *ICS '04: Proceedings of the 18th Annual International Conference on Supercomputing*, pages 196–206. ACM Press, 2004.

[21] J. Seng, D. Tullsen, and G. Cai. Power-Sensitive Multithreaded Architecture. In *ICCD '00: Proceedings of the 2000 IEEE International Conference on Computer Design*, page 199. IEEE Computer Society, 2000.

[22] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder. Automatically Characterizing Large Scale Program Behavior. In *ASPLOS-X: Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 45–57, 2002.

[23] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *HPCA '02: Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, page 17, Washington, DC, USA, Feb. 2002. IEEE Computer Society.

[24] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *ISCA '03: Proceedings of the 30th International Symposium on Computer Architecture*, Apr. 2003.

[25] A. Snavely, D. Tullsen, and G. Voelker. Symbiotic Jobscheduling with Priorities for a Simultaneous Multithreading Processor, June 2002.

[26] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. The Case for Lifetime Reliability-Aware Microprocessors. In *ISCA '04: Proceedings of the 31st International Symposium on Computer Architecture*, page 276. IEEE Computer Society, 2004.

[27] M. Tremblay. High Performance Throughput Computing (Niagara). keynote presentation for *31st ISCA '04: 31st International Symposium on Computer Architecture*. Sun Microsystems, June 2004.

[28] D. Tullsen and J. Brown. Handling Long-Latency Loads in a Simultaneous Multithreaded Processor. In *MICRO-34: Proceedings of the 34th International Symposium on Microarchitecture*, 2001.

[29] D. Tullsen, S. Eggers, and H. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. In *ISCA '95: Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 392–403, June 1995.