Appears in: *Proceedings of the 7th International Symposium on High-Performance Computer Architecture, Monterrey, Mexico, January 2001*

Dynamic Thermal Management for High-Performance Microprocessors

David Brooks
Department of Electrical Engineering
Princeton University
dbrooks@ee.princeton.edu

Margaret Martonosi
Department of Electrical Engineering
Princeton University
mrm@ee.princeton.edu

Abstract

With the increasing clock rate and transistor count of today's microprocessors, power dissipation is becoming a critical component of system design complexity. Thermal and power-delivery issues are becoming especially critical for high-performance computing systems.

In this work, we investigate dynamic thermal management as a technique to control CPU power dissipation. With the increasing usage of clock gating techniques, the average power dissipation typically seen by common applications is becoming much less than the chip's rated maximum power dissipation. However, system designers still must design thermal heat sinks to withstand the worst-case scenario.

We define and investigate the major components of any dynamic thermal management scheme. Specifically we explore the tradeoffs between several mechanisms for responding to periods of thermal trauma and we consider the effects of hardware and software implementations. With appropriate dynamic thermal management, the CPU can be designed for a much lower maximum power rating, with minimal performance impact for typical applications.

1 Introduction

Today's highest performance microprocessors contain on the order of one hundred million transistors with this number continuing to grow with Moore's Law. The majority of the transistors on these chips exist to extract ILP and to reduce memory access times for a range of common applications; unfortunately, the performance benefits of many of these techniques are gradually becoming overshadowed by increased design complexity and power dissipation. As we move towards billion transistor microprocessors, the growing power budgets of these chips must be addressed at all levels of the design cycle.

The system complexity associated with increased power dissipation can be divided into two main areas. First, there is the cost and complexity of designing thermal packaging which can adequately cool the processor. It is estimated that after exceeding 35-40W, additional power dissipation increases the total cost per CPU chip by more than $1/W [23]. The second major source of design complexity involves power delivery, specifically the on-chip decoupling capacitances required by the power distribution network.

Unfortunately, these cooling techniques must be designed to withstand the *maximum* possible power dissipation of the microprocessor, even if these cases rarely occur in typical applications. For example, while the Alpha 21264 processor is rated as having a maximum power dissipation of 95W when running "max-power" benchmarks, the average power dissipation was found to be only 72W for typical applications [10]. The increased use of clock gating and other power management techniques that target average power dissipation will expand this gap even further in future processors. This disparity between the *maximum* possible power dissipation and the *typical* power dissipation suggests dynamic thermal management techniques to ensure that the processor does not reach these maximum power dissipation levels. That is, we seek to explore scenarios where the cooling apparatus is designed for a wattage less than the true maximum power, and dynamic CPU approaches guarantee that this designed-for level is never exceeded during a program run.

With many industrial designers predicting that power delivery and dissipation will be the primary limiters of performance and integration of future high-end processors, we feel that some form of dynamic thermal management will eventually be seen as a performance optimization, enabling larger chips to be built which would otherwise not be feasible [10, 23, 2, 12]. If die area and the number of transistor per chip become constrained by power density, techniques that can constrain the maximum possible power dissipation could allow designers to include more transistors per chip than would otherwise be possible, thus leading to increased performance.

In this work, we define and examine the generic mechanisms inherent in dynamic thermal management (DTM) schemes. Section 2 provides an overview and background on dynamic thermal management. We explore and compare the potential for hardware and software-based implementations of several dynamic thermal management schemes. Section 3 discusses the methodology used in the remainder of the paper. We then break thermal management systems into three components: triggers, responses, and initiation policies, and discuss each of them in Sections 4, 5, and 6 respectively. The core of any DTM system is how it responds to a thermal emergency (e.g. frequency scaling, execution throttling, etc.). While this paper provides data on a number of possible responses, we feel that further work may identify even more effective ones. Thus, Section 7 outlines a methodology for identifying promising new response techniques by comparing power and performance correlations. Finally, Section 8 offers conclusions.

2 Dynamic Thermal Management: Overview and Strategies

This paper explores policies and mechanisms for implementing dynamic thermal management in current and future high-end CPUs. As we use it, the term dynamic thermal management refers to a range of possible hardware and software strategies which work dynamically, at run-time, to control a chip's operating temperature. Traditionally, the packaging and fans for a CPU or computer system were designed to be able to maintain a safe operating temperature even when the chip was dissipating the maximum power possible for a sustained period of time, and therefore generating the highest amount of thermal energy. This worst-case thermal scenario is highly unlikely, however, and thus such worst-case packaging is often expensive overkill. DTM allows packaging engineers to design systems for a target sustained thermal value that is much closer to average-case for real benchmarks. If a particular workload operates above this point for sustained periods, a DTM response will work to reduce chip temperature. In essence, DTM allows designers to focus on average, rather than worst-case, thermal conditions in their designs. Until now, techniques developed to reduce *average* CPU power have garnered only moderate interest among the designers of high-end CPUs because thermal considerations, rather than battery life, were their primary concern. Therefore, in addition to reducing packaging costs, DTM improves the leverage of techniques such as clock gating designed to reduce average power [23, 3].

The key goals of DTM can be stated as follows: (i) to provide inexpensive hardware or software responses, (ii) that reliably reduce power, (iii) while impacting performance as little as possible. Voltage and frequency scaling are two methods for DTM that have been implemented in current chips [15, 24]. Unfortunately, little work has been done on quantifying the impact of voltage or frequency scaling on application performance. This paper seeks to address this need, while also proposing other microarchitectural approaches for implementing DTM. We also propose a methodology based on performance and power correlations for seeking out new DTM responses.

2.1 Overview and Terminology

In this paper, we are primarily concerned with reducing the maximum power dissipation of the processor. From a pure hardware point of view, the maximum power dissipation occurs when all of the structures within the processor are active with maximum switching activity. However, mutual exclusions in the underlying control structures make this scenario impossible. In reality, the maximum power dissipation is constrained by the software program that can maximize the usage and switching activity of the hardware. Special max-power benchmarks can be written to maximize the switching activity of the processor. These benchmarks are often quite esoteric, perform no meaningful computation, and dissipate higher power than "real" programs. Thus, DTM techniques could be used solely to target power levels seen in maximum power benchmarks and would rarely be invoked during the course of typical applications. In this paper, we also consider more aggressive DTM designs which seek to further reduce the amount of cooling hardware necessary in machines. In Section 5.2 we discuss the tradeoffs between cooling hardware and performance loss in more detail.

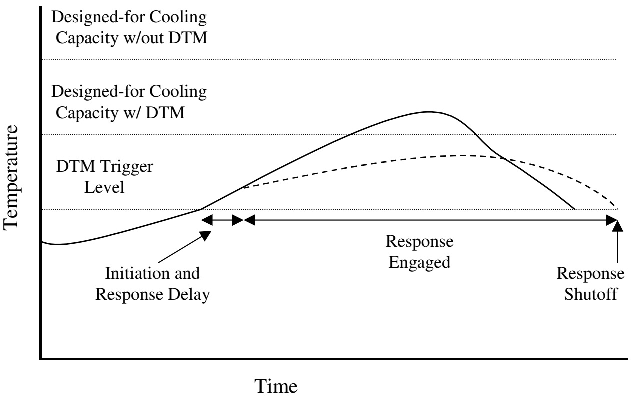

Figure 1. Overview of Dynamic Thermal Management (DTM) technique.

Figure 1 offers a motivating example of how dynamic thermal management (DTM) can work. This figure plots chip temperature versus time (in cycles). In this figure, there are three horizontal dashed lines. The top-most line shows the designed-for cooling capacity of the machine without DTM. The second line shows that the cooling capacity could be reduced if dynamic techniques were implemented, because DTM reduces the effective maximum power dissipation of the machine. Finally, the lowest horizontal line shows the DTM trigger level. This is the temperature at which the DTM techniques are engaged.

Figure 1 has two curves which show chip temperature for some sequence of code being executed on the machine. The upper, solid curve is executed on the machine without

DTM, and the lower, dotted curve is executed on a machine that has implemented DTM. Both curves are the same until the DTM trigger level is exceeded. At this point, after a small delay to engage the response, the curves diverge. In the uppermost curve the chip temperature slowly increases and then falls back below the trigger level. The lower curve shows how DTM would affect the same sequence of code. In this case, the DTM response is able to reduce the power dissipation and hence the chip temperature; the temperature never exceeds the designed-for cooling capacity. Eventually, the temperature decreases (as in the non-DTM curve), and the response is dis-engaged with some performance delay relative to the non-DTM curve.

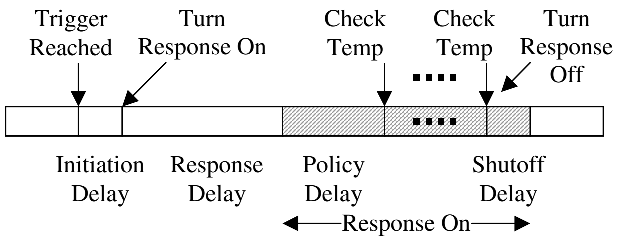

Figure 2. Mechanisms for Dynamic Thermal Management.

Figure 2 breaks down a DTM instance into several components. First, DTM is *triggered*. The triggering event may be a thermal sensor, a power estimator, or other gauge which indicates when DTM is needed. Once the trigger goes off, there is some *initiation delay* while, for example, an operating system interrupt and handler are invoked to interpret the triggering event. Once the handler has been executed, some DTM *response* begins. For example, possible responses include voltage or frequency scaling [17], or some of the microarchitectural ideas we discuss in later sections. Depending on the type of response chosen, there may be some delay inherent in invoking it; we refer to this time as *response delay*. Once the response is in effect, the next issue concerns when to turn it off. Turning the response off as soon as the temperature dips below the threshold may be unwise; temperature may fluctuate around the threshold and warrant keeping the response turned on. We use the term *policy delay* to refer to the number of cycles we wait before checking to see if the temperature has dipped below the triggering level. Finally, once the DTM system has determined that the response should be turned off, there is often a *shutoff delay* while, for example, the voltage or frequency is readjusted.

Implementing an effective DTM system, therefore, involves several key design choices which we consider throughout the remainder of this paper:

- Selecting simple and effective triggers (Section 4),
- Identifying useful response mechanisms (Section 5),
- Developing policies for when to turn responses on and off (Section 6).

2.2 Background and Related Work

Some dynamic thermal management techniques have previously been explored. For example, the G3 and G4 PowerPC microprocessors from Motorola include a thermal temperature sensor in hardware and an interrupt capability to notify software of when a particular temperature has been reached [19, 21]. The processor also includes an instruction cache throttling mechanism that allows the processor's fetch bandwidth to be reduced when the CPU reaches a temperature limit.

The Transmeta Crusoe processor includes "LongRun" technology which dynamically adjusts CPU supply voltage and frequency to reduce power consumption [24]. While voltage and frequency tuning are quite effective at reducing power consumption (power scales linearly with clock frequency and with the square of the supply voltage), the delay in triggering these responses is necessarily higher than with microarchitectural techniques that are more localized. One of the goals of this paper is to provide an overall view for the tradeoffs between initiation delay, response delay, and performance overhead for a number of techniques including both previously published techniques as well as ones newly-proposed in this paper.

In addition to the fairly-recent Crusoe work, the ACPI (Advanced Configuration and Power Interface) specification likewise works to have hardware and software cooperate to manage power dynamically [1]. Unlike our work or those described above, ACPI is very coarse-grained. That is, power management in ACPI involves actions like turning on or off I/O devices or managing multiple batteries. Our work seeks to provide a much more fine-grained solution to thermal problems within the CPU itself. Recent work, including our own, has operated on different thursts to explore this domain [4, 22, 13]. These papers and our own each focus on distinct classes of processor architectures. Rohou and Smith have also considered using temperature feedback to guide the operating system in controlling CPU activity on a per-application basis [20].

Finally, we also note that the work by both Motorola and Transmeta is mainly geared toward improving battery life in portable machines. Our work, in contrast, has thermal packaging in high-end CPUs as its main thrust. This context is more performance sensitive than is power management for laptops. Our overall goal is to guarantee much lower worst-case power consumption, so that cheaper packaging can be used, with as little impact on performance as possible.

Parameter	Value
Processor Core	
LSQ/RUU size	40/80 instructions
Fetch Queue Size	8 instructions
Fetch width	4 instructions/cycle
Decode width	4 instructions/cycle
Issue width	4 instructions/cycle (out-of-order)
Commit width	4 instructions/cycle (in-order)
Functional Units	4 INT ALUs, 1 INT MUL/DIV
	2 FP ADD/MUL/SQRT
Branch Prediction	
Branch Predictor	Combined, Bimodal 4K table
	2-Level 1K table, 10bit history
	4K chooser, 1024-entry, 2-way
RAS	32-entry
Mispredict penalty	7 cycles
Memory Hierarchy	
L1 D-cache	64K, 2-way (LRU)
	32B blocks, 1 cycle latency
L1 I-cache	64K, 2-way (LRU)
	32B blocks, 1 cycle latency
L2	Unified, 2M, 4-way (LRU)
	32B blocks, 12-cycle latency
Memory	100 cycles
TLBs	128-entry, 30-cycle miss latency

Table 1. Baseline Configuration of Simulated Processor

3 Methodology

We have developed an architectural-level power modeling tool called Wattch [5]. Wattch provides power modeling extensions to the SimpleScalar architecture simulator [7]. Wattch's power modeling infrastructure is based on parameterized power models of common structures present in modern superscalar microprocessors. Per-cycle power estimates are generated by scaling these power models with hardware access counts and activity factors from the performance simulator.

3.1 Simulation Model Parameters

Unless stated otherwise, our results in this paper model a processor with the configuration parameters shown in Table 1. For technology parameters, we use the process parameters for a .35um process at 600MHz. We use Wattch's aggressive clock gating style for all results. This models power scaling which is linear with the number of active ports on any particular unit.

3.2 Benchmark Applications

We evaluate our ideas on programs from the SPECint95 and SPECfp95 benchmark suites. SPEC95 programs are representative of a wide mix of current integer and floating-point codes. We have compiled the benchmarks for the Alpha instruction set using the Compaq Alpha *cc* compiler with the following optimization options as specified by the SPEC Makefile: -migrate -std1 -O5 -ifo -non_shared. For each program, we simulate 200M instructions.

3.3 Power vs. Temperature

Wattch provides per-cycle power estimates, but one challenge in this research has been translating these power estimates into chip-level temperature variations. The most accurate approach would be to develop a model for the chip packaging and heat sink in a microprocessor. We are currently discussing such models with packaging engineers, but have abstracted them for the research presented here. We use the average power over a suitably large chunk of cycles (10k, 100k, and 1M) as a proxy for temperature [18].

4 Dynamic Thermal Management: Trigger Mechanisms

Any dynamic response technique requires a trigger mechanism to engage the response during program execution. In this section, we consider two aspects of the trigger mechanism. First, we describe several possible trigger mechanisms for dynamic thermal management. Second, we discuss the rationale for determining an appropriate trigger limit to use in the DTM system. Sections 5 and 6 discuss the other key parts of the system: response techniques and initiation mechanisms.

4.1 Trigger Mechanisms

For our experimental setup we use an abstraction of chip temperature by using the moving average of power dissipation for the last 10,000 cycles of the processor's operation. This trigger mechanism is similar to an on-chip temperature sensor. We will discuss the details of the temperature sensor as well as several other trigger mechanisms that could be used as abstractions for temperature.

- **Temperature Sensors for Thermal Feedback**

  In the PowerPC dynamic thermal management system, thermal feedback from an on-chip temperature sensor is used as the trigger mechanism [21]. In the proposed scheme, the temperature sensor compares the junction temperature with a user programmable threshold. If the value is exceeded an interrupt is triggered allowing the operating system to invoke a response mechanism. This is the basic trigger mechanism that we evaluate in Section 5 with a variety of response mechanisms.

- **On-chip Activity Counters**

Another possible source of information regarding the current chip temperature is through the use of activity monitors or on-chip performance counters [8]. These devices record "activity factors" for various structures within the processor and thus provide a gauge of much work is being done and the correspondingly the thermal state of the machine.

- **Dynamic profiling analysis**

  The runtime system of the machine can be responsible for determining when the application or user-behavior does not require the full resources of the computing system and trigger a response. For example, operating systems often provide a wait process which is entered when there is no work to be performed, or address access information can be used to determine when the processor is idling [16].

  In addition, certain real-time and user-interactive applications inherently set certain acceptable performance levels. These types of applications would allow dynamic thermal management to occur when the specified rate is exceeded [9].

- **Compile-time trigger requirements**

  Static analysis at compile time can be used to estimate the performance of applications. We feel that in a similar manner, the compiler could estimate the high-power code segments and insert instructions specifying that DTM triggers should occur. In EPIC or VLIW where more of the parallelism is exposed by the compiler, this method would be more fruitful.

Comparing the viability of various trigger mechanisms is a topic for future research in this area. Relying exclusively on chip temperature sensors may have some drawbacks. First, the temperature sensor only approximates the average chip temperature; multiple sensors may be needed on large chips. Second, there may be hysteresis between the temperature reading and the actual temperature. Pure hardware solutions also do not provide information about the workload; a combination of temperature sensors, activity counters, and software analysis may more effective than any of the techniques taken alone. For the results presented in this paper, we use an abstracted trigger mechanism based on interrupts when modeled power reaches a a pre-set trigger threshold. This approximates the situation of a CPU with a single temperature sensor.

4.2 Thermal Trigger and Emergency Settings

The second decision that must be made within the trigger mechanism is the pre-set trigger threshold. We will define a "thermal trigger" to be the temperature threshold at which the trigger mechanism initiates the response mechanism to begin to cool the processor's temperature. A "thermal emergency" is a second temperature threshold set to a higher level and is used as a gauge of how successful the response mechanism was in dealing with the increase in temperature. Except where noted, in our simulation environment thermal triggers and emergencies occur if the moving average of full chip power dissipation for the past 10,000 cycles exceeds the pre-set trigger and emergency wattage values. Likewise there are also triggers that indicate the CPU has returned to a safe temperature. At these trigger points, the CPU can begin returning to normal operation.

Benchmark	Cycles in Emergency	Average Power
go	1.0%	22.7W
cc1	1.6%	21.6W
ijpeg	32.7%	24.3W
li	50.9%	24.8W
vortex	61.6%	24.6W
su2cor	70.5%	25.1W
tomcatv	96.1%	25.5W
fpppp	98.4%	32.9W

Table 2. Baseline Configuration of Simulated Processor

In the next two sections we present analysis for the case where the response is triggered when the 10k moving average exceeds 24W and a full-fledged thermal emergency is considered to occur when the 10k moving average exceeds 25W. In Section 5.2, we consider the effects of varying the trigger level and the 10k cycle thermal window, but for the rest of the results we will use these values. Table 2 shows the percent of cycles that were above the thermal emergency threshold for the baseline system without DTM with the 24W trigger. There are three main categories of applications; the remainder of our charts will be sorted as follows:

- Mild Thermal Demands: The first two benchmarks have less than 10% of their cycles in thermal emergencies with average powers much less than the emergency level.

- Intensive Thermal Demands: The second group of four benchmarks ranges from 32% to 96%. *Tomcatv* fell into this class because its average power is only just above the emergency level.

- Extreme Thermal Demands: *Fpppp* is the extreme case in which 98% of the cycles exceeded the thermal threshold and the average power was 7W above the threshold.

We selected this trigger setting and this set of applications so we could observe the impact of DTM in a range of scenarios with varying thermal demands. We have neglected *compress* and *m88ksim* from the analysis because neither application had any cycles exceeding the chosen emergency point.

5 Dynamic Thermal Management: Response Mechanisms

In this section we consider the second basic mechanism within a dynamic thermal management architecture. The goal of designing a good DTM scheme is to reduce power with as small a performance loss as possible. A key part to realizing this goal is the response mechanism that throttles power dissipation in the system.

In this work, we consider five response mechanisms. Three of these are microarchitectural responses: I-cache-toggling, speculation control by restricting the number of unresolved branches, and decode bandwidth throttling (similar to Motorola's I-cache throttling). We also consider clock frequency scaling and a combination of clock frequency scaling and voltage scaling.

- **Clock Frequency Scaling**

  Clock frequency scaling essentially trades a linear performance loss for a linear power savings. While in principle clock frequency scaling is trivial to implement, there may be delays incurred when changing clock rates. Furthermore, communicating with synchronous devices on the system bus may become more complicated.

- **Voltage and Frequency Scaling**

  Transmeta's LongRun technology performs dynamic clock frequency scaling along with dynamic voltage scaling to reduce power dissipation when necessary [24]. Obviously this requires detailed timing analysis and careful attention to circuit design choices [6]. Furthermore, as future process technologies scale to lower base supply voltages, dynamic voltage scaling may become more difficult. This is especially true when standby leakage currents become important. Leakage currents are directly related to the supply voltage; lowering the supply voltage to dynamically reduce dynamic power would have a corresponding increase in the standby leakage current.

- **Decode Throttling**

  The PowerPC G3 microprocessor uses a microarchitectural level dynamic thermal management technique called instruction cache throttling to restrict the flow of instructions to the processor core [21]. This scheme relies on clock gating to reduce power dissipation as the flow of instructions is restricted. As motivation for selecting I-cache throttling instead of clock frequency scaling, the authors cite the difficulty in implementing dynamic clock control for the on-chip PLL as well as the fact the chip's L2 cache interface operates at a different clock rate from the chip's core.

- **Speculation Control**

  Speculation control is similar to Manne's work on speculative pipeline gating based on branch confidence estimation [11]. However, with the method proposed here, instead of basing the speculation control on branch confidence as in [11], we arbitrarily restrict the amount of speculation in the pipeline whenever a thermal trigger level is reached. To implement this, a counter is incremented whenever a branch is decoded and decremented whenever a branch resolves. If the counter exceeds a software-set limit, the decode stage stalls until enough branches have been resolved. The infrastructure for restricting the number of resolved branches is most likely already in place in most processors, since they limit the number of branches in the pipeline to restrict the additional state required for each active branch.

- **I-cache Toggling**

  We also propose a microarchitectural response technique called *I-cache toggling*. This response involves disabling the instruction fetch unit (I-cache and branch prediction) and using the instruction fetch queue to feed the pipeline. The fetch unit can be disabled every cycle, every other cycle, or at any specified interval as specified by the interrupt call.

Obviously other techniques, or combinations of techniques, could be used as the response mechanism. In Section 7, we discuss a systematic methodology for determining new response techniques.

Both the trigger and the various response mechanisms that have been discussed could be programmable, allowing system designers to specify thermal management levels based on the amount of heat-sink technology in the system. For example, more expensive high-end server systems could have higher trigger limits and allow more unresolved branches, while cheaper low-end desktop systems would have lower trigger limits corresponding to their smaller heat-sinks. In addition, the individual response mechanisms allow a variation in the amount of throttling to be performed.

5.1 Response Mechanism Results

We use two metrics to evaluate the DTM schemes. First, the scheme should reduce the number of cycles in which the processor's temperature exceeds the thermal emergency threshold. The second metric that we use is the overall performance loss that the DTM technique incurs. Since the schemes that we evaluate rely on microarchitectural as well as frequency scaling techniques, we consider total execution time as our performance metric.

We present analysis for the case where the response is triggered when the 10k moving average exceeds 24W and a full-fledged thermal emergency is considered to occur when the 10k moving average exceeds 25W. We also assume here that the various responses are initiated by a 250-cycle interrupt from the operating system in a manner similar to that of the PowerPC, but in Section 6 we consider additional hardware support which improves the performance of thermal management by eliminating this operating system overhead.

For the response that includes voltage scaling we assume a 10 microsecond delay to switch frequencies and a 20 microsecond response delay to switch voltages. During this delay we assume that the processor is stalled; this is consistent with the delay that Transmeta reports when switching between frequency and voltage states [14]. For the scaling techniques we set the policy delay to be 15 microseconds; in Section 6 we consider extending this delay to reduce the performance overhead of initiating the response. Finally, we assume that the processor voltage scales proportionally to what is reported in [14] for each frequency level. For example, when we scale frequency down by 10%, voltage is scaled down by 4.2% for the combined voltage and frequency scaling technique.

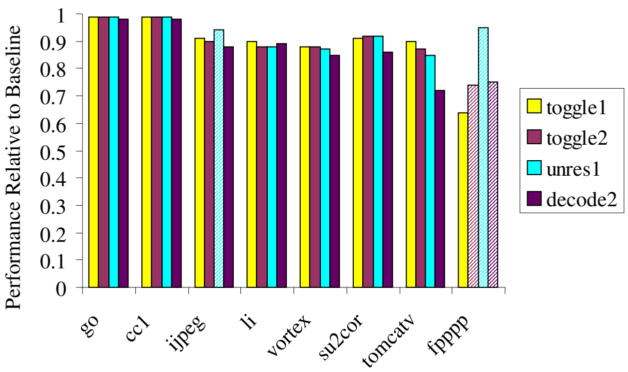

Figure 3. Reduction in performance compared to baseline for microarchitecture techniques.

Figure 3 shows the overall program performance reduction from the baseline for the microarchitectural techniques. Figure 4 shows the same results for the frequency/voltage scaling based techniques. Within these figures the first two sets of bars correspond to the benchmarks with mild thermal demands. The next five bars have intensive thermal demands. Finally, we show the *fpppp*, the extreme case benchmark.

Within Figure 3 there are four bars for each benchmark.

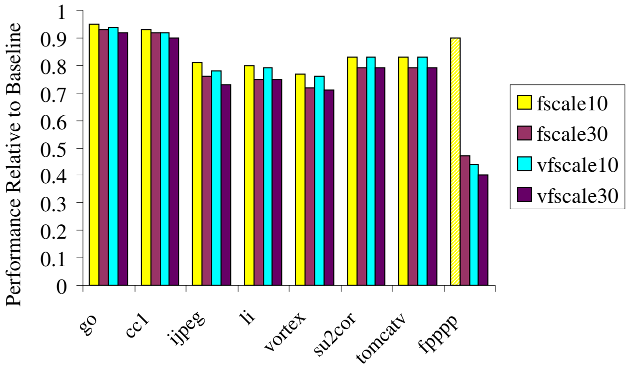

Figure 4. Reduction in performance compared to baseline for frequency/voltage scaling techniques.

The first two bars correspond to I-cache toggling; *toggle1* is the case where the I-cache is disabled every cycle during the response, *toggle2* corresponds to the case in which it is disabled every other cycle. The third bar labeled *unres1* indicates that the maximum number of unresolved branches allowed in the pipeline is restricted to one before the decode stage is stalled. The final bar *decode2* indicates that the decode bandwidth is reduced by two instructions per cycle respectively. (We have considered additional settings for the above parameters, but to save space, we have selected the parameters that performed the best.) Within both Figure 3 and 4, bars which are cross-hatched (for example, *fpppp*'s *toggle2*, *unres1*, and *decode2* bars) indicate that the thermal emergencies were not entirely reduced for this configuration. Figure 4 also has four bars per benchmark. The first two bars correspond to scaling down the frequency by 30% and 10%. The last two bars correspond to scaling both the frequency and the voltage by 30% and 10%.

For many of the benchmarks, all of the techniques were able to entirely eliminate the thermal emergencies in the machine at this trigger level. DTM was not successful in entirely removing thermal emergencies with *ijpeg* with the *unres1* technique and *fpppp* with three of the microarchitectural techniques and *fscale10*.

For the benchmarks with mild thermal demands, the microarchitectural level techniques incurred an average performance penalty was 2%; the voltage and frequency scaling techniques had a 7% drop. For the benchmarks with intensive thermal demands, the reduction in thermal emergencies incurred a 12% performance penalty for the microarchitectural techniques and a 22% performance penalty for the scaling techniques. Only *toggle1*, *fscale30%*, *vfscale10*, and *vfscale30* were effective at reducing the number of thermal emergencies with *fpppp*; this came at over a 35% performance penalty.

Clearly the performance degradation of DTM at this trigger/emergency level is significant for the applications with large thermal demands. The performance degradation from these techniques can be broken down into two components. The first component is the performance drop due to invocation of the techniques. This includes the overhead of the operating system interrupt calls and the time needed to dynamically adjust the frequency and voltage scaling of the system. The second component is the IPC drop of the microarchitectural techniques or the frequency degradation penalty of the scaling techniques. For example with *su2cor* and the *unres1* trigger, 26% of the performance degradation was due to the interrupt overhead to engage and disengage the trigger. The remainder of the performance drop was the IPC degradation due to restricting the number of unresolved branches. As expected, the trigger overhead with frequency and voltage scaling techniques is much higher; over 70% of the performance loss is incurred due to the interrupt calls and overhead in adjusting the clock rate with frequency scaling and over 75% with combined voltage and frequency scaling.

These trends tend to hold across the benchmarks and across the different styles of responses. There are two major reasons for the larger invocation overhead of the frequency and voltage scaling techniques. First, the overhead of frequency and voltage scaling is significantly higher than the microarchitectural techniques. Second, because of variations in application behavior which cause changes in the thermal behavior of the system these policies may be enabled or disabled many times during program execution. This is especially true when DTM mechanisms are in place to regulate temperature. Obviously for these applications, the large invocation overhead is magnified. In the next section we consider additional hardware and other techniques that can reduce the performance overhead of trigger engagement. The results also show that there is room for application specific selection of response techniques; certain response techniques perform much better than others on an individual benchmark basis.

5.2 Thermal Trigger and Emergency Settings

In Figure 5 we consider an idealized version of the *vfscale30* policy that has no initiation delay. This figure shows the the percent performance loss relative to the total execution time of the baseline system for DTM while varying the thermal trigger settings ranging from 20-34W. For example, *fpppp* runs 27% slower with a trigger of 20W than it does with no DTM, but its max power without DTM exceeds 40W in some cases. This performance penalty is incurred by the response mechanism; in this case the response is a version of frequency scaling. When the trigger is set at a conservative range (above 30W for these benchmarks), most of the benchmarks see very little performance degradation. Even with the most conservative approach, dynamic thermal management allows the chip's maximum power rat-

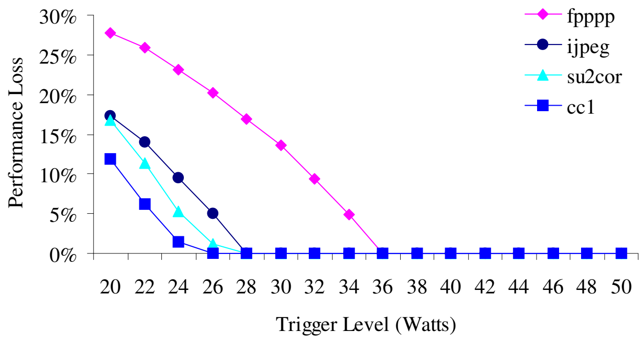

Figure 5. Performance Loss at various trigger levels. Higher trigger levels (30-50 Watts) offer less packaging savings, but have nearly zero performance impact. More aggressive trigger settings (25-30 Watts) begin to show modest performance impact. 50W is the max power for the modeled chip.

ing to be reduced considerably. In this design, the maximum power was around 50W; with DTM this could be easily reduced to 35-40W.

A more aggressive design would set the trigger somewhere around 25W for these applications. Being more aggressive in the trigger setting allows for more significant packaging savings, about $1 per watt per CPU chip. But this savings may come at the price of reduced performance for some applications. Thus, a key goal of this paper is to propose streamlined mechanisms for DTM that offer the best possible performance.

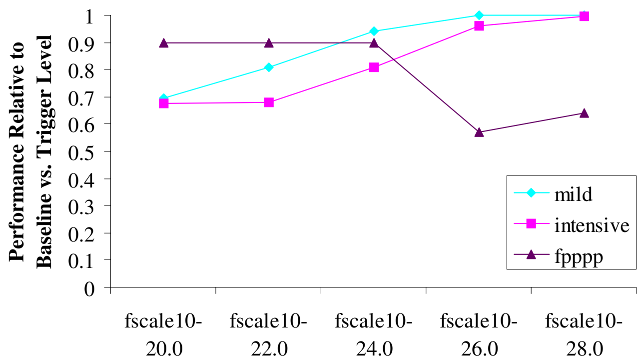

Figure 6. Performance Loss at various trigger levels for the *fscale10* response technique.

Now we consider the effect of the trigger value with our standard (including all delays) *fscale10* technique. Figures 6 and 7 show the effects of varying the trigger level for the *fscale10* technique. Each data point shows the performance and number of thermal emergencies relative to the baseline configuration without DTM at the specified trigger level; the level that we consider to be an emergency is always set to be 1W above the trigger level. From Figure 6 we can see

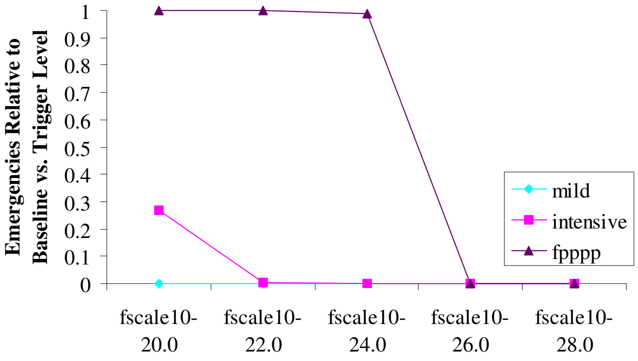

Figure 7. Emergency Reduction at various trigger levels for the *fscale10* response technique.

that for the set of mild and intensive benchmarks, performance degrades further as we set more aggressive trigger levels. However, from Figure 7 we see that the machine does not exceed the thermal emergency threshold until we reach a trigger of 20W. *Fpppp* performs quite differently – at trigger levels between 20-24W, the number of thermal emergencies has not been reduced at all; the *fscale10* policy is continuously engaged leading to a constant 10% performance penalty. However, at 26W and 28W the *fscale10* policy begins to be effective. At the 26W trigger level there is a corresponding drop in performance as we start to see the effect of the trigger being engaged and disengaged during execution. At 28W and upwards, this performance penalty diminishes.

We have seen similar patterns with the other voltage and frequency scaling techniques as well as with the microarchitectural techniques. Overall, we see that the choice of the trigger level is an important lever for system designers to use when deciding whether to trade off performance for some of the most extreme benchmarks such as *fpppp* against the amount of cooling hardware to build into the system.

6 Dynamic Thermal Management: Initiation Mechanisms

In Section 5 we consider a variety of dynamic response mechanisms. In that section, we assume an implementation where the operating system calls an interrupt handler to invoke the dynamic response mechanism which incurs significant overhead. To mitigate this overhead, we consider two modifications to the initiation mechanism of DTM. First, we consider additional hardware support in the microarchitecture to remove the interrupt overhead. Second, we modified the policy delay to allow the response mechanism to remain engaged for longer periods of time, better amortizing the cost of the trigger's response delay over the program run.

6.1 Hardware Support for Initiating Responses

Eliminating interrupt call overhead is the obvious benefit from additional hardware support. However, avoiding interrupt handling also allows more fine-grained control of the response scheme. This reduces the performance overhead of DTM because the performance-limiting response will only be engaged when it is needed. Finally, more fine-grained control of the response mechanism could have a benefit on reducing the number of cycles with thermal emergencies, because the mechanism will be engaged faster.

To eliminate the trigger overhead, the trigger mechanism must be directly integrated into the microarchitecture. For example, the temperature sensor or hardware activity counter could generate a signal indicating that the trigger limit has been exceeded and this signal could be sent to microarchitectural state machines which would engage the trigger. The operating system would only need to program internal registers at the beginning of the application's execution to adjust the amount of throttling that the response should use. To evaluate the effects of this additional hardware we have simulated the microarchitectural response techniques with a 0-cycle initiation delay; this assumes that the 250-cycle interrupt overhead can be removed.

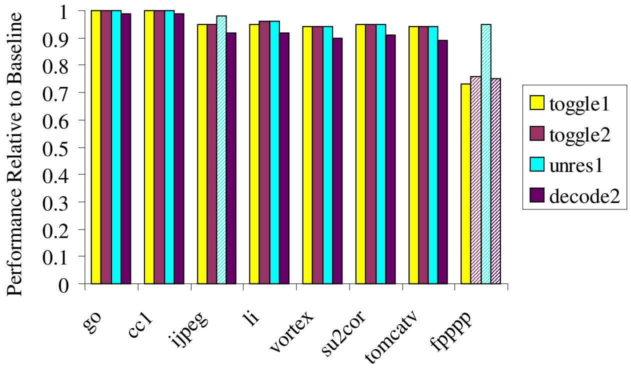

Figure 8. Reduction in performance compared to baseline.

Figure 8 shows the results for the microarchitectural response mechanisms assuming that the trigger mechanism is integrated into the microarchitecture. The reduction in number of thermal emergencies is unchanged. However, there was a reduction in performance penalty. For the mildly intensive four benchmarks, the performance penalty was on average 5%; this compares to a 7% performance hit without the hardware support. For the next group of four benchmarks with more intensive thermal demands, the performance reduction was 13% compared to 16% with OS overhead. Since *fpppp* spends a large amount of time with the triggers engaged, speeding up the interrupt overhead had a small effect on the performance using this scheme.

6.2 Policy and Thermal Window Effects on Voltage/Frequency Scaling

In the previous section, we considered the use of hardware support to reduce the overhead of initiating the response mechanism. This overhead is even larger for the voltage and frequency scaling techniques. The majority of the time required to initiate these techniques is spent scaling the frequency and internal voltage of the processor to a new level. Since this overhead is not related to the operating system, reducing the interrupt time will only have a small effect on performance. In this section we consider two techniques to reduce this delay. First, we consider increasing the policy delay, or the amount of time that the mechanism is enabled before it is eligible to be disabled. Increasing the policy delay allows the response and shutoff overhead to be amortized over a larger portion of the run. On the other hand, if the policy delay is too long, the response will be engaged during unnecessary stretches of program execution. The second technique we consider is using a larger thermal window to estimate the temperature of the chip. For all of the previous results, we have used a window of 10K cycles. In this section, we consider increasing this window to be 100K cycles. This has the effect of smoothing out short thermal spikes which could unnecessarily cause the response to be triggered. For the more coarse-grained frequency and voltage scaling techniques, we would like to minimize these situations.

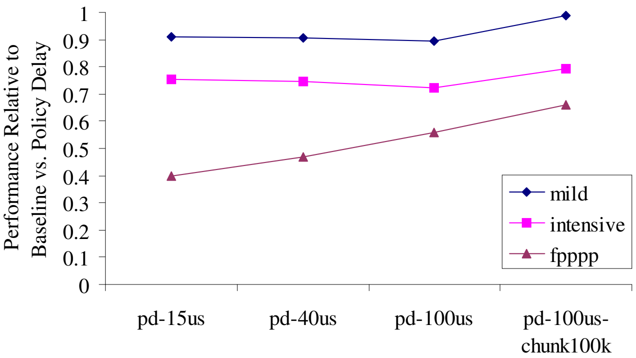

Figure 9. Reduction in performance compared to baseline.

We consider varying the policy delay with values of 15 microseconds, 40 microseconds, and 100 microseconds. We have chosen 40 microseconds because it is the combined response and shutoff delays of frequency+voltage scaling response mechanism. Finally, we show the effect of increasing the thermal window from 10K cycles to 100K cycles with a policy delay of 100 microseconds. Figure 9 shows the the performance effect of the techniques when using the *vfscale30*. In this figure, the first three data points report the performance relative to the baseline while varying the policy delay; the final point shows the performance as the thermal window is increased to 100K cycles.

From this Figure 9 we see that there was very little effect on performance for the mild and intensive benchmark suite; in fact, there was a slight degradation in performance as we increase the policy delay. This is because although the initiation overhead was decreased, the amount of time spent with frequency and voltage scaling engaged increased. On the other hand, *fpppp* had a substantial performance improvement with increased policy delay. For this benchmark, the performance loss to the baseline decreased from 60% with 15 microsecond policy delay to 44% with 100 microsecond policy delay. Finally, we see that increasing the thermal window had a positive effect on all three classes of applications. When moving from the 10K cycle window to the 100K cycle window the performance loss decreased to 34% for *fpppp*.

For the benchmarks with intensive thermal demands, the performance loss decreased to 20%. On the other hand, we found that increasing the size of the thermal window had a much smaller (1-2%) performance benefit for the microarchitecural techniques. Since these techniques are much more fine-grained in nature, they suffer less from shorter thermal transients.

We have found that the initiation mechanism is a key factor to the performance degradation of DTM. We have investigated two techniques which show promise for reducing the performance overhead. Future work could address additional techniques to reduce this overhead either through more efficient methods to initiate the responses or smarter techniques to enable and disable responses.

7 Method for Identifying DTM Responses

In this work we have compared the benefits of dynamic thermal management via several microarchitectural techniques as well as clock frequency and voltage scaling. In considering other schemes for thermal management, we would like to develop a more systematic approach to identifying potential techniques.

We propose here a method based on correlation. That is, we wish to find levers that reduce power with a less-than-proportional reduction in performance. We have performed simulations using Wattch to correlate power dissipation with other processor statistics such as instruction fetch rate, branch prediction accuracy, data and instruction cache hit rates, execution bandwidth, and IPC. We use this method to isolate certain processor statistics which track more closely with power than with IPC.

We collected the average power and performance statistical data for fixed chunks of 10,000 cycles. These statistics were then correlated with each other after the simulation completed. An example of the correlation data for the average of our benchmark suite is shown in Table 3. The first

Correlation	Fetch Rate	BPred Rate	DC Hit Rate	IC Hit Rate	Exec BW
Power vs.	0.82	0.37	0.40	0.50	0.83
IPC vs.	0.77	0.61	0.25	0.48	0.81
Difference	0.05	-0.24	0.15	0.02	0.02

Table 3. Correlation Data for Average of Benchmarks

line of this table shows the correlation between processor power dissipation and instruction fetch rate (avg. number of instructions fetched per cycle), branch prediction accuracy, cache hit rates, and execution bandwidth (committed + mis-speculated instructions/cycle). As expected, power correlates very strongly with execution bandwidth. Instruction fetch bandwidth correlates also correlates strongly with power. Branch direction prediction accuracy, a secondary indicator of application performance, also correlates with power but to a lesser degree. Data and instruction-cache hit rates correlate slightly more than branch predictor accuracy with power.

The second line of Table 3 shows the correlation between IPC and the processor statistics. From this table, we see execution bandwidth, branch predictor accuracy, and fetch bandwidth correlate the most with performance.

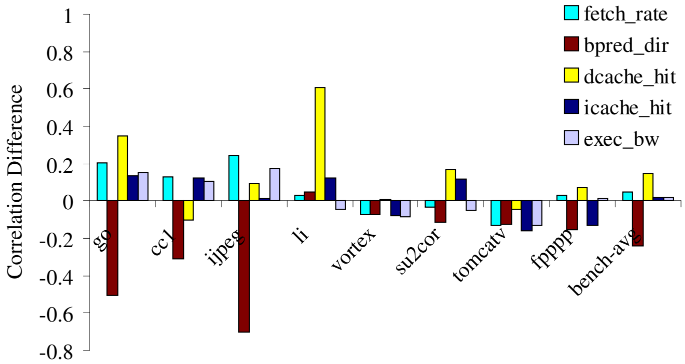

Figure 10. Correlation between power and several performance statistics.

Figure 10 plots power correlation minus IPC correlation for each point for the individual benchmarks. Thus, a positive data point in this graph corresponds to a case where power dissipation is more strongly correlated with the metric (eg fetch rate) than IPC is. Looking for possible DTM responses with strong power correlations lets us seek out “wasted work” that may lead to good power reductions with minimal performance impact. This may reveal strategies that would be most useful for dynamic thermal management.

This data reveals some interesting trends. For example, for almost all of the benchmarks, branch predictor accuracy correlated much more with performance than with power. On other other hand, cache hit rates and instruction fetch bandwidth correlated more with power than with IPC for many of the benchmarks. Execution bandwidth correlates more with power than with IPC for four of the benchmarks. This lends support to our decision to evaluate I-cache toggling and speculation control as methods for dynamic thermal management. We plan future work that will broaden the types of microarchitectural response mechanisms that we investigate with correlation analysis.

8 Conclusion

We have proposed and evaluated the benefits of using dynamic thermal management to reduce the cooling system costs of CPUs. From this initial research effort, we have drawn several conclusions which we feel can help guide future research in this area.

- **Trigger Selection:** Dynamic thermal management allows arbitrary tradeoffs between performance and savings in cooling hardware. Conservative target selections can still lead to significant cost improvements with essentially zero performance impact, because the trigger point is rarely reached for many applications.

- **Designers Can Focus on Average Power:** In addition, DTM makes other techniques targeting average power more interesting to the designers of high-end CPUs. Effective DTM makes average power the metric of interest even for high-end CPU designers, since packages need no longer be designed for worst-case power. With DTM, lowering average CPU power will reduce the trigger value needed for a particular level of performance, and thus will reduce packaging costs.

- **Trigger Activation Time is Significant:** Not unexpectedly, the triggering delay is a key factor in the performance overhead of DTM. We have found that more fine-grained control of the trigger mechanism is especially important in the context that we consider: reducing thermal traumas in high-performance CPUs. Unfortunately, our data show that some of the most promising techniques in DTM today, such as voltage or frequency scaling, are typically implemented with very high activation delays. These lead to significant performance overheads across most applications.

- **Lightweight Policies Are Effective:** More lightweight, fine-grained policies, such as the microarchitectural techniques we have discussed, often allows the temperature to stay close to the target level with a small performance penalty. In addition, the fine-grained policies are less affected by rapid fluctuations in the temperature.

- **Methodology for Identification of Future Techniques:** Because of these growing opportunities for microarchitectural DTM techniques, we have also proposed a methodology for evaluating new DTM approaches. This mechanism correlates power and performance, and looks for "low-hanging fruit"; that is, our correlators look for techniques that can cut power by significantly more than they hurt performance. Identifying these sorts of wasted work, particularly on an application-specific basis, appears to be a promising way of discovering new microarchitectural DTM techniques in the future.

9 Acknowledgement

This research was supported in part by an NSF ITR grant, a donation from Intel, and an IBM University Partnership award. In addition, David Brooks is supported by an NSF Graduate Research Fellowship and a Princeton University Gordon Wu Fellowship.

References

[1] Advanced Configuration and Power Interface. http://www.teleport.com/ acpi/.

[2] S. Borkar. Design Challenges of Technology Scaling. *IEEE Micro*, July-August 1999.

[3] D. Brooks and M. Martonosi. Dynamically exploiting narrow width operands to improve processor power and performance. In *Proc. of the 5th Int'l Symp. on High-Performance Computer Architecture*, Jan. 1999.

[4] D. Brooks and M. Martonosi. Adaptive thermal management for high-performance microprocessors. In *Workshop on Complexity Effective Design 2000 at ISCA27*, June 2000.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, June 2000.

[6] T. Burd, T. Pering, A. Stratkos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *ISSCC Digest of Technical Papers*, pages 294–295, 2000.

[7] D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. *Computer Architecture News*, pages 13–25, June 1997.

[8] C. Georgiou and S. Kirkpatrick and T. Larson. Variable Chip-clocking Mechanism. US Patent 5,189,314, 1993.

[9] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC variation in workloads with externally specified rates to reduce power consumption. In *Complexity-Effective Design at ISCA27*, June 2000.

[10] M. Gowan, L. Biro, and D. Jackson. Power considerations in the design of the Alpha 21264 microprocessor. In *35th Design Automation Conference*, 1998.

[11] D. Grunwald, A. Klauser, S. Manne, and A. Pleszkun. Confidence estimation for speculation control. In *Proc. of the 25th Int'l Symp. on Computer Architecture*, pages 122–31, June 1998.

[12] L. Gwennap. Power issues may limit future CPUs. *Microprocessor Report*, August 1996.

[13] W. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A framework for dynamic energy efficiency and temperature management. In *Proc. of the 33rd Int'l Symp. on Microarchitecture*, Dec. 2000.

[14] Marc Fleischmann. Crusoe Power Management: Cutting x86 Operating Power Through LongRun. Embedded Processor Forum, June 2000.

[15] J. Montanaro et al. A 160-MHz, 32-b, 0.5W CMOS RISC microprocessor. *Digital Technical Journal*, 9(2):49–62, 1996.

[16] O. Ikeda. Power Saving Control System for a Computer System. US Patent 5,504,908, 1996.

[17] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings of International Symposium on Low Power Electronics and Design*, August 1998.

[18] Prof. Matt Krane. Materials Science Department, Purdue University. Thermal Packaging Models. Personal communication, Dec. 1999.

[19] P. Reed et al. 250 MHz 5W RISC microprocessor with on-chip L2 cache controller. *Digest of Technical Papers - IEEE International Solid-State Circuits Conference*, 40:412, 1997.

[20] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *2nd Workshop on Feedback-Directed Optimization*, Nov. 1999.

[21] H. Sanchez et al. Thermal management system for high performance powerpc microprocessors. *Digest of Papers - COMPCON - IEEE Computer Society International Conference*, page 325, 1997.

[22] J. Seng, D. Tullsen, and G. Cai. Power-sensitive multithreaded architecture. In *International Conference on Computer Design 2000*, Sep. 2000.

[23] V. Tiwari et al. Reducing power in high-performance microprocessors. In *35th Design Automation Conference*, 1998.

[24] Transmeta Corp. The Technology Behind the Crusoe Processor Whitepaper, 2000.