Temperature-Aware Microarchitecture: Extended Discussion and Results

UNIV. OF VIRGINIA DEPT. OF COMPUTER SCIENCE TECH. REPORT CS-2003-08*

APRIL 2003

†Kevin Skadron, ‡Mircea R. Stan, ‡Wei Huang, †Sivakumar Velusamy,
†Karthik Sankaranarayanan, and †David Tarjan†

†*Dept. of Computer Science,* ‡*Dept. of Electrical and Computer Engineering*
University of Virginia, Charlottesville, VA
{*skadron,siva,karthick,dtarjan*}*@cs.virginia.edu,* {*mircea,wh6p*}*@virginia.edu*

Abstract

With power density and hence cooling costs rising exponentially, processor packaging can no longer be designed for the worst case, and there is an urgent need for runtime processor-level techniques that can regulate operating temperature when the package's capacity is exceeded. Evaluating such techniques, however, requires a thermal model that is practical for architectural studies.

This paper expands upon the discussion and results that were presented in our conference paper [43]. It describes HotSpot*, an accurate yet fast model based on an equivalent circuit of thermal resistances and capacitances that correspond to microarchitecture blocks and essential aspects of the thermal package. Validation was performed using finite-element simulation. The paper also introduces several effective methods for dynamic thermal management (DTM): "temperature-tracking" frequency scaling, localized toggling, and migrating computation to spare hardware units. Modeling temperature at the microarchitecture level also shows that power metrics are poor predictors of temperature, that sensor imprecision has a substantial impact on the performance of DTM, and that the inclusion of lateral resistances for thermal diffusion is important for accuracy.*

1. Introduction

In recent years, power density in microprocessors has doubled every three years [7, 27], and this rate is expected to increase within one to two generations as feature sizes and frequencies scale faster than operating voltages [40]. Because energy consumed by the microprocessor is converted into heat, the corresponding exponential rise in heat density is creating vast difficulties in reliability and manufacturing costs. At any power-dissipation level, heat being generated must be removed from the surface of the microprocessor die, and for all but the lowest-power designs today, these cooling solutions have become expensive. For high-performance processors, cooling solutions are rising at $1–3 or more per watt of heat dissipated [7, 18], meaning that cooling costs are rising exponentially and threaten the computer industry's ability to deploy new systems.

Power-aware design alone has failed to stem this tide, requiring *temperature*-aware design at all system levels, including the processor architecture. Temperature-aware design will make use of power-management techniques, but probably in ways that are different from those used to improve battery life or regulate peak power. Localized heating occurs much faster than chip-wide heating; since power dissipation is spatially non-uniform across the chip, this leads to "hot spots" and spatial gradients that can cause timing errors or even physical damage. These effects evolve over time scales of hundreds of microseconds or milliseconds. This means that power-management techniques, in order to be used for thermal management, must directly target the spatial and temporal behavior of operating temperature. In fact, *many low-power techniques have little or no effect on operating temperature*, because they do not reduce power density in *hot spots*, or because they only

*This report is an extended version of a paper appearing in the 30th International Symposium on Computer Architecture (ISCA-30), San Diego, CA, June 2003.

†This work was conducted while David Tarjan visited U.Va. during his diploma program at the Swiss Federal Institute of Technology Zürich.

reclaim slack and do not reduce power and temperature when no slack is present. Temperature-aware design is therefore a distinct albeit related area of study.

Temperature-specific design techniques to date have mostly focused on the thermal package (heat sink, fan, etc.). If the package is designed for worst-case power dissipation, they must be designed for the most severe hot spot that could arise, which is prohibitively expensive. Yet these worst-case scenarios are rare: the majority of applications, especially for the desktop, do not induce sufficient power dissipation to produce the worst-case temperatures. A package designed for the worst case is excessive.

To reduce packaging cost without unnecessarily limiting performance, it has been suggested [8, 18, 20] that the package should be designed for the worst *typical* application. Any applications that dissipate more heat than this cheaper package can manage should engage an alternative, *runtime* thermal-management technique (*dynamic thermal management* or DTM). Since typical high-power applications still operate 20% or more below the worst case [18], this can lead to dramatic savings. This is the philosophy behind the thermal design of the Intel Pentium 4 [18]. It uses a thermal package designed for a *typical* high-power application, reducing the package's cooling requirement by 20% and its cost accordingly. Should operating temperature ever exceed a safe temperature, the clock is stopped (we refer to this as *global clock gating*) until the temperature returns to a safe zone. This protects against both timing errors and physical damage that might result from sustained high-power operation, from operation at higher-than-expected ambient temperatures, or from some failure in the package. As long as the threshold temperature that stops the clock (the *trigger threshold*) is based on the hottest temperature in the system, this approach successfully regulates temperature. This technique is similar to the "fetch toggling" technique proposed by Brooks and Martonosi [8], in which instruction fetch is halted when the trigger threshold is exceeded.

The Need for Architecture-Level Thermal Management. These *chip-level* hardware techniques illustrate both the benefits and challenges of runtime thermal management: while they can substantially reduce cooling costs and still allow typical applications to run at peak performance, these techniques also reduce performance for any applications that exceed the thermal design point. Such performance losses can be substantial with chip-wide techniques like global clock gating, with a 27% slowdown for our hottest application, *art*.

Instead of using chip-level thermal-management techniques, we argue that the microarchitecture has an essential role to play. *The microarchitecture is unique in its ability to use runtime knowledge of application behavior and the current thermal status of different units of the chip to adjust execution and distribute the workload in order to control thermal behavior.* In this paper, we show that architecture-level thermal modeling exposes architectural techniques that regulate temperature with lower performance cost than chip-wide techniques by exploiting instruction-level parallelism (ILP). For example, one of the better techniques we found—with only an 8% slowdown—was a "local toggling" scheme that varies the rate at which only the hot unit (typically the integer register file) can be accessed. ILP helps mitigate the impact of reduced bandwidth to that unit while other units continue at full speed.

Architectural solutions do not of course preclude software or chip-level thermal-management techniques. Temperature-aware task scheduling, like that proposed by Rohou and Smith [34], can certainly reduce the need to engage any kind of runtime hardware technique, but there will always exist workloads whose operating temperature cannot successfully be managed by software. Chip-level fail-safe techniques will probably remain the best way to manage temperature when thermal stress becomes extreme, for example when the ambient temperature rises above specifications or when some part of the package fails (for example, the heat sink falls off). But all these techniques are synergistic, and only architectural techniques have detailed temperature information about hot spots and temperature gradients that can be combined with dynamic information about instruction-level parallelism in order to precisely regulate temperature while minimizing performance loss.

The Need for Architecture-Level Thermal Modeling. Some architectural techniques have already been proposed [8, 20, 26, 36, 41], so there is clearly interest in this topic within the architecture field. To accurately characterize current and future thermal stress, temporal and spatial non-uniformities, and application-dependent behavior—let alone evaluate architectural techniques for managing thermal effects—a model of temperature is needed. *Yet the architecture community is currently lacking reliable and practical tools for thermal modeling*. As we show in this paper, the current technique of estimating thermal behavior from some kind of average of power dissipation is highly unreliable. This has led prior researchers to incorrectly estimate the performance impact of proposed thermal-management techniques and even to target their thermal-management techniques at areas of the chip that are not hot spots.

An effective architecture-level thermal model must be simple enough to allow architects to reason about thermal effects and tradeoffs; detailed enough to model runtime changes in temperature within different functional units; and yet computationally efficient and portable for use in a variety of architecture simulators. Even software-level thermal-management techniques will benefit from thermal models. Finally, the model should be flexible enough to easily extend to novel computing systems that may be of interest from a temperature-aware standpoint. Examples include like graphics and network processors, MEMS, and processors constructed with nanoscale materials.

Contributions. This paper illustrates the importance of thermal modeling; proposes a *compact, dynamic, and portable* thermal model for convenient use at the architecture level; uses this model to show that hot spots typically occur at the granularity of architecture-level blocks, and that power-based metrics are not well correlated with temperature; and discusses some remaining needs for further improving the community's ability to evaluate temperature-aware techniques. Our model—which we call *HotSpot*—is publicly available at http://lava.cs.virginia.edu/hotspot. Using this model, we evaluate a variety of DTM techniques. The most effective technique is "temperature-tracking" dynamic frequency scaling: timing errors due to hotspots can be eliminated with an average slowdown of 2%, and, if frequency can be changed without stalling computation, less than 1%. For temperature thresholds where preventing physical damage is also a concern, using a spare register file and migrating computation between the register files in response to heating is the best, with an average slowdown of 5–7.5%. Local toggling and an overhead-free voltage scaling technique performed almost as well, both with slowdowns of about 8%. All our experiments include the effects of sensor imprecision, which significantly handicaps runtime thermal management.

Because thermal constraints are becoming so severe, we expect that temperature-aware computing will be a rich area for research, drawing from the fields of architecture, circuit design, compilers, operating systems, packaging, and thermodynamics. We hope that this paper provides a foundation that stimulates and helps architects to pursue this topic with the same vigor that they have applied to low power.

This paper is an extended version of our conference paper [43], allowing us to present an expanded discussion of various issues and present expanded results. The rest of this paper is organized as follows. The next section provides further background and related work. Then Section 3 describes our proposed model, its derivation and validation, and shows the importance of modeling temperature rather than power. Section 4 presents several novel thermal-management techniques and explores the role of thermal-sensor non-idealities on thermal management, with Section 5 describing our experimental setup, issues concerning initial temperatures, and the time-varying behavior of some programs. Section 6 compares the various thermal-management techniques' ability to regulate temperature and discusses some of the results in further detail, and Section 7 concludes the paper.

2. Background and Related Work

2.1. Cooling Challenges

Power Density. Power densities have been rising despite reduced feature sizes and operating voltages, because the number of transistors has been doubling every eighteen months and operating frequencies have been doubling every two years [40]. Power densities are actually expected to rise faster in future technologies, because difficulties in controlling noise margins mean that operating voltage (V_{dd}) can no longer scale as quickly as it has: for 130nm and beyond, the 2001 International Technology Roadmap for Semiconductors (ITRS) [40] projects very little change in V_{dd}. The rising heat generated by these rising power densities creates a number of problems, because both soft errors and aging increase exponentially with temperature. The most fundamental is thermal stress. At sufficiently high temperatures, transistors can fail to switch properly (this can lead to soft or hard errors), many failure mechanisms are significantly accelerated (e.g. electromigration) which leads to an overall decrease in reliability, and both the die and the package can even suffer permanent damage. Yet to maintain the traditional rate of performance improvement that is often associated with Moore's Law, clock rates must continue to double every two years. Since carrier mobility is inversely proportional to temperature, operating temperatures cannot rise and may even need to *decrease* in future generations for high performance microprocessors. The ITRS actually projects that the maximum junction temperature decreases from 95°C for 180nm to 85°C for 130nm and beyond. Spatial temperature gradients exacerbate this problem, because the clock speed must typically be designed for the hottest spot on the chip, and information from our industrial partners suggests that temperatures can vary by 30 degrees or more under typical operating conditions. Such spatial non-uniformities arise both because different units on the chip exhibit different power densities, and because localized heating occurs much faster than chip wide heating due to the slow rate of lateral heat propagation. Yet another temperature-related effect is leakage, which is exponentially increasing with operating temperature. Increasing leakage currents in turn dissipate additional heat, which in the extreme can even lead to a destructive vicious cycle called thermal runaway.

Reliability. From a reliability standpoint, manufacturers must ensure that their microprocessors will meet timing margins and will operate for some reasonable number of years (typically 5–10 years) under a range of ambient conditions and altitudes, under a range of thermal gradients across the chip and its package, and even operate correctly in the face of a failure in the thermal package, such as the heat sink's becoming detached or a fan failure. These thermal requirements are typically specified in terms of a maximum operating temperature beyond which timing errors may begin to occur, the reliability is significantly reduced, and even physical damage might occur.

Timing and hence operating frequency are dependent on temperature because carrier mobility (both holes and electrons) decreases with increasing temperatures. This is the reason why some supercomputer families use refrigeration for increased performance, even if the costs associated with such complex cooling solutions are impractical for general purpose computers.

The typical reliability model for such a thermal requirement based on a maximum temperature is based on the Arrhenius equation, which links exponentially the mean time to failure (MTTF) to the absolute temperature (T):

$$MTTF = Ae^{-\frac{E_a}{k \cdot T}} \tag{1}$$

where A is an empirical constant, E_a is the so-called activation energy and k is Boltzmann's constant. Lately the Arrhenius equation has been viewed as inadequate because it does not include other reliability effects like thermal cycling and spatial thermal gradients. But the exponential dependence of reliability on temperature is yet another reason to favor lower operating temperatures, and is why Viswanath *et al.* [48] observe that reducing temperature by even 10–15° can improve the lifespan of a device by as much as a factor of two. Industry sources tell us that an operating temperatures as high as 100° are typically safe from a reliability standpoint, and so at the operating temperatures specified by the ITRS for high-performance processors, the main concern will be hotspot-induced timing errors.

Costs of Cooling Solutions. Transistor counts and frequencies are rising because the information-technology industry currently demands that performance double approximately every 18 months. The consequent increase in CPU heat dissipation is inexorable, and improved package design and manufacturing techniques to reduce their cost are the mainstay of coping with this heat. Yet these packages are now becoming prohibitively expensive, cutting profits in markets that are already experiencing declining profit margins [27].

Architecture techniques can play an important role by allowing the package to be designed for the power dissipation of a typical application rather than a worst-case application, and by exploiting instruction-level parallelism to allow extreme applications to still achieve reasonable performance even though they exceed the capacity of the package.

2.2. Related Work

Non-Architectural Techniques. A wealth of work has been conducted to design new packages that provide greater heat-removal capacity, to arrange circuit boards to improve airflow, and to model heating at the circuit and board (but not architecture) levels. Compact models are the most common way to model these effects, although computational fluid dynamics using finite-element modeling is often performed when the flow of air or a liquid is considered. An excellent survey of these modeling techniques is given by Sabry in [35]. Batty *et al.* [4], Cheng and Kang [13], Koval and Farmaga [23], Székely *et al.* [32, 45], and Torki and Ciontu [46] all describe techniques for modeling localized heating within a chip due to different power densities of various blocks, but none of these tools are easily adapted to architectural exploration for a variety of reasons. Architectural modeling typically precludes direct thermal-response measurements, *e.g.* [45, 46], the use of analytic power models obviates the need for joint electro-thermal modeling, *e.g.* [45], and these techiques typically depend on low-level VLSI netlists and structural implementation details or only give steady-state solutions.

In addition to the design of new, higher-capacity packages, quiet fans, and the choice of materials for circuit boards and other components, recent work at the packaging level has given a great deal of consideration to liquid cooling to achieve greater thermal conductivity (but cost and reliability are concerns) [1]; heat pipes to spread or conduct the heat to a location with better airflow (especially attractive for small form factors like laptops), *e.g.* [48]; and to high-thermal-mass packages that can absorb large quantities of heat without raising the temperature of the chip—this heat can then be removed during periods of low computation activity or DVS can be engaged if necessary, *e.g.* [12].

Architectural Techniques. Despite the long-standing concern about thermal effects, only a few studies have been published in the architecture field, presumably because power itself has only become a major concern to architects within the past five years or so, and because no good models existed that architects could use to evaluate thermal-management techniques.

Gunther *et al.* [18] describe the thermal design approach for the Pentium 4, where thermal management is accomplished via global clock gating. Lim *et al.* [26] propose a heterogeneous dual-pipeline processor for mobile devices in which the standard execution core is augmented by a low-power, single-issue, in-order pipeline that shares the fetch engine, register files, and execution units but deactivates out-of-order components like the renamer and issue queues. The low-power pipeline is primarily intended for applications that can tolerate low performance and hence is very effective at saving energy, but this technique is also potentially effective whenever the primary pipeline overheats. This work used Tempest [15], which does model temperature directly, but only at the chip level, and no sensor effects are modeled. Performance degradation is not reported, only energy-delay product.

Huang *et al.* [20] deploy a sequence of four power-reducing techniques—a filter instruction cache, DVS, sub-banking for the data cache, and if necessary, global clock gating—to produce an increasingly strong response as temperature approaches the maximum allowed temperature. Brooks and Martonosi [8] compared several stand-alone techniques for thermal management: frequency scaling, voltage and frequency scaling, fetch toggling (halting fetch for some period of time, which is similar to the Pentium 4's global clock gating), decode throttling (varying the number of instructions that can be decoded per cycle [36]), and speculation control (varying the number of in-flight branches to reduce wasteful mis-speculated execution [28]). Brooks and Martonosi also point out the value of having a direct microarchitectural thermal trigger that does not require a trap to the operating system and its associated latency. They find that only fetch toggling and aggressive DVS are effective, and they report performance penalties in the same range as found by the Huang group. Unfortunately, while these papers stimulated much interest, no temperature models of any kind were available at the time these papers were written, so both use chip-wide power dissipation averaged over a moving window as a proxy for temperature. As we show in Section 3.7, this value does not track temperature reliably. A further problem is that, because no model of localized heating was available at the time, it was unknown which units on the processor ran the hottest, so some of the proposed techniques do not reduce power density in those areas which industry feedback and our own simulations suggest to be the main hot spots, namely the register files, load-store queue, and execution units. For example, we found that the low-power cache techniques are not effective, because they do not reduce power density in these other hot units.

Our prior work [41] proposed a simple model for tracking temperature on a per-unit level, and feedback control to modify the Brooks fetch-toggling algorithm to respond gradually, showing a 65% reduction in performance penalty compared to the all-or-nothing approach. The only other thermal model of which we are aware is TEMPEST, developed by Dhodapkar *et al.* [15]. TEMPEST also models temperature directly using an equivalent RC circuit, but contains only a single RC pair for the entire chip, giving no localized information. A chip-wide model allows some exploration of chip-wide techniques like DVS, fetch toggling, and the Pentium 4's global clock gating, but not more localized techniques, and does not capture the effects of hot spots or changing chip layout. No prior work in the architecture field accounts for imprecision due to sensor noise and placement.

This paper shows the importance of a more detailed thermal model that includes localized heating, thermal diffusion, and coupling with the thermal package, and uses this model to evaluate a variety of techniques for DTM.

3. Thermal Modeling at the Architecture Level

3.1. Using an Equivalent RC Circuit to Model Temperature.

There exists a well-known duality [24] between heat transfer and electrical phenomena, summarized in Table 1. Heat flow can be described as a "current" passing through a thermal resistance, leading to a temperature difference analogous to a "voltage". Thermal capacitance is also necessary for modeling transient behavior, to capture the delay before a change in power results in the temperature's reaching steady state. Lumped values of thermal R and C can be computed to represent the heat flow among units and from each unit to the thermal package. The thermal Rs and Cs together lead to exponential rise and fall times characterized by thermal RC time constants analogous to the electrical RC time constants. The rationale behind this duality is that current and heat flow are described by exactly the same differential equations for a potential difference. In the thermal-design community, these equivalent circuits are called *compact models*, and *dynamic compact models* if they include thermal capacitors. This duality provides a convenient basis for an architecture-level thermal model. For a microarchitectural unit, heat conduction to the thermal package and to neighboring units are the dominant mechanisms that determine the temperature.

Thermal quantity	unit	Electrical quantity	unit
P, Heat flow, power	W	I, Current flow	A
T, Temperature difference	K	V, Voltage	V
R_{th}, Thermal resistance	K/W	R, Electrical resistance	$\Omega = V/A$
C_{th}, Thermal mass, capacitance	J/K	C, Electrical capacitance	$F = A/V$
$\tau_{th} = R_{th} \cdot C_{th}$, Thermal RC constant	s	$\tau = R \cdot C$, Electrical RC constant	s

Table 1. Duality between thermal and electrical quantities

3.2. A Parameterized, BICI, Dynamic Compact Model for Microarchitecture Studies

For the kinds of studies we propose, the compact model must have the following properties. It must track temperatures at the granularity of individual microarchitectural units, so the equivalent RC circuit must have at least one node for each unit. It must be parameterized, in the sense that a new compact model is automatically generated for different microarchitectures; and portable, making it easy to use with a range of power/performance simulators. It must be able to solve the RC-circuit's differential equations quickly. It must be calibrated so that simulated temperatures can be expected to correspond to what would be observed in real hardware. Finally, it must be *BICI*, that is, boundary- and initial-condition independent: the thermal model component values should not depend on initial temperatures or the particular configuration being studied. The HotSpot model we have developed meets all these conditions. It is a simple library that provides an interface for specifying some basic information about the package and for specifying any floorplan that corresponds to the architectural blocks' layout. HotSpot then generates the equivalent RC circuit automatically, and, supplied with power dissipations over any chosen time step, computes temperatures at the center of each block of interest. The model is BICI by construction since the component values are derived from material, physical, and geometric values. Because the HotBlocks RC network is not obtained from full solutions of the time-dependent heat-conduction equation for every possible architecture, there may be boundary and initial condition sets that would lead to inaccurate results. But for the range of reasonable architectural floorplans and the short time scales of architecture simulations, the parameterized derivation of the chip model ensures that the model is indeed BICI for practical purposes if not in a formal sense. In other words, architects can rely on the model to be independent of boundary and initial conditions for purposes or architectural simulation.

Chips today are typically packaged with the die placed against a spreader plate, often made of aluminum, copper, or some other highly conductive material, which is in turn placed against a heat sink of aluminum or copper that is cooled by a fan. This is the configuration modeled by HotSpot. A typical example is shown in Figure 1. Low-power/low-cost chips often omit the heat spreader and sometimes even the heat sink; and mobile devices often use heat pipes and other packaging that avoid the weight and size of a heat sink. These extensions remain areas for future work.

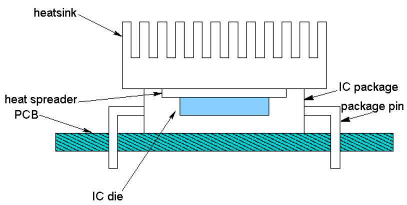


Figure 1. Side view of a typical package.

The equivalent circuit—see Figure 2 for an example—is designed to have a direct and intuitive correspondence to the physical structure of a chip and its thermal package. The RC model therefore consists of three vertical, conductive layers for the die, heat spreader, and heat sink, and a fourth vertical, convective layer for the sink-to-air interface. The die layer is divided into blocks that correspond to the microarchitectural blocks of interest and their floorplan. For simplicity, the example in Figure 2 depicts a die floorplan of just three blocks, whereas a realistic model would have 10-20 or possibly even more. The spreader is divided into five blocks: one that corresponds to the area right under the die (R_{sp}), and four trapezoids corresponding to the periphery that is not covered by the die. In a similar way, the sink is divided into five blocks: one corresponding to the area right under the spreader (R_{hs}); and four trapezoids for the periphery. Finally, the convective heat transfer from the package to the air is represented by a single thermal resistance ($R_{convection}$). Air is assumed to be at

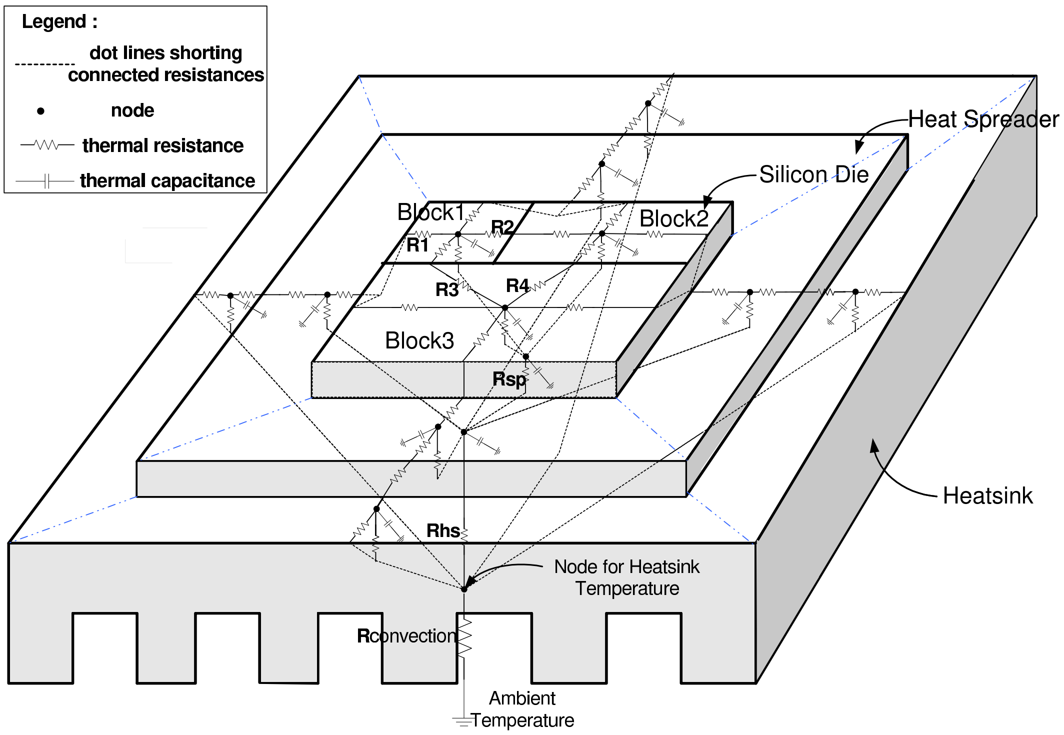


Figure 2. Example HotSpot RC model for a floorplan with three architectural units, a heat spreader, and a heat sink. The RC model consists of three layers: die, heat spreader, and heat sink. Each layer consists of a vertical RC pair from the center of each block down to the next layer and a lateral RC pair from the center of each block to the center of each edge.

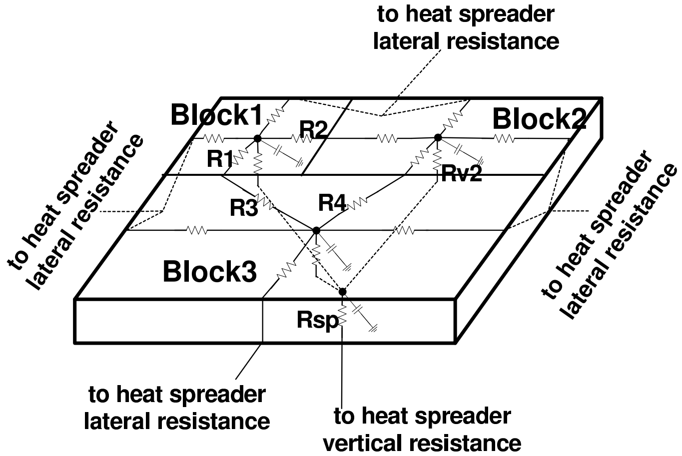


Figure 3. The RC model for just the die layer.

a fixed ambient temperature, which is often assumed in thermal design to be 45°C [27] (this is not the room ambient, but the temperature inside the computer "box"). Because the perspective in Figure 2 makes it somewhat difficult to distinguish vertical and lateral Rs, Figure 3 shows the RC model for only Rs in the die layer. To clarify how the components of Figure 2 correspond to a typical chip and its package, Figure 1 shows a side-view cross-section of a die in its package. Note that we currently neglect the small amount of heat flowing into the die's insulating ceramic cap and into the I/O pins, and from there into the circuit board, etc. We also neglect the interface materials between the die, spreader, and sink. These are all further areas for future work.

For the die, spreader, and sink layers, the RC model consists of a vertical model and a lateral model. The vertical model captures heat flow from one layer to the next, moving from the die through the package and eventually into the air. For example, R_{v2} in Figure 3 accounts for heat flow from Block 2 into the heat spreader. The lateral model captures heat diffusion between adjacent blocks within a layer, and from the edge of one layer into the periphery of the next area (*e.g.*, $R1$ accounts for heat spread from the edge of Block 1 into the spreader, while $R2$ accounts for heat spread from the edge of Block 1 into the rest of the chip). At each time step in the dynamic simulation, the power dissipated in each unit of the die is modeled as a current source (not shown) at the node in the center of that block.

3.3. Deriving the Model

In this section we sketch how the values of R and C are computed. The derivation is chiefly based upon the fact that thermal resistance is proportional to the thickness of the material and inversely proportional to the cross-sectional area across which the heat is being transferred:

$$R = \frac{t}{k \cdot A} \tag{2}$$

where k is the thermal conductivity of the material per unit volume, $100W/m \cdot K$ for silicon and $400W/m \cdot K$ for copper at 85°C. Thermal capacitance, on the other hand, is proportional to both thickness and area:

$$C = c \cdot t \cdot A \tag{3}$$

where c is the thermal capacitance per unit volume, $1.75 \times 10^6 J/m^3 \cdot K$ for silicon and $3.55 \times 10^6 J/m^3 \cdot K$ for copper. Note that HotSpot requires a scaling factor to be applied to the capacitors to account for some simplifications in our lumped model relative to a full, distributed RC model, and these are described below. These factors are analytical values derived from physical properties.

Typical chip thicknesses are in the range of 0.3–0.9mm, with some recent wafers we have in our laboratory measuring 0.45-0.5mm; this paper studies a "thinned" chip of 0.5mm thickness. Thinning adds to the cost of production but reduces localized hotspots by reducing the amount of relatively higher-resistance silicon and getting the heat-generating areas closer to the higher-conductivity metal package.

In addition to the basic derivations above, lateral resistances must account for *spreading resistance* between blocks of different aspect ratios, and the vertical resistance of the heat sink must account for *constriction resistance* from the heat-sink base into the fins [25]. Spreading resistance accounts for the increased heat flow from a small area to a large one, and vice-versa for constriction resistance. These calculations are entirely automated within HotSpot. For example, R1 in Figure 2 accounts for spreading from Block 1 into the surrounding spreader area; and R2 accounts for spreading from Block 1 into the rest of the die. When blocks are not evenly aligned, as in the case of Block 3 relative to Blocks 1 and 2, multiple resistors in parallel are used, like R3 and R4. The equivalent resistance of R3 and R4 together equals the spreading resistance from Block 3 into the rest of the chip, and R3 and R4 are then determined in proportion to the shared edge with Blocks 1 and 2 respectively.

To clarify how spreading/constriction resistances are computed, consider the two adjacent blocks, Block 1 and Block 2, in Figure 4. The lengths are $L1$ and $L2$ respectively. The chip thickness is t. Now we calculate the lateral resistance R21, which is the thermal resistance from the center of Block 2 to the shared edge of Blocks 1 and 2. In this case, we can consider the heat is constricted from Block 1 to Block 2 via the surface areas defined by $L1 \cdot t$ and $L2 \cdot t$. The constriction thermal resistance can be calculated by assuming the heat source area to be $L1 \cdot t$, the silicon bulk area that accepts the heat to be $L2 \cdot t$, and the thickness of the bulk to be $W2/2$. With these values found, we can calculate the spreading/constriction resistance based on the formulas given in [25]. The resistance is a spreading one if the lateral area of the source is smaller than the bulk lateral area, and it is a constriction one on the other hand.

When calculating the chip lateral resistances, each block is assumed to present a thermal resistance on each of its four sides. These resistances are effectively the constriction thermal resistances from outside the block into the four sides of the

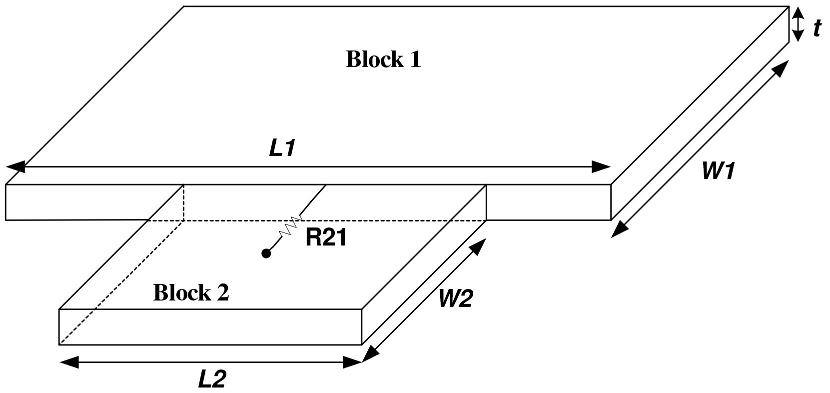


Figure 4. Example to illustrate spreading resistance.

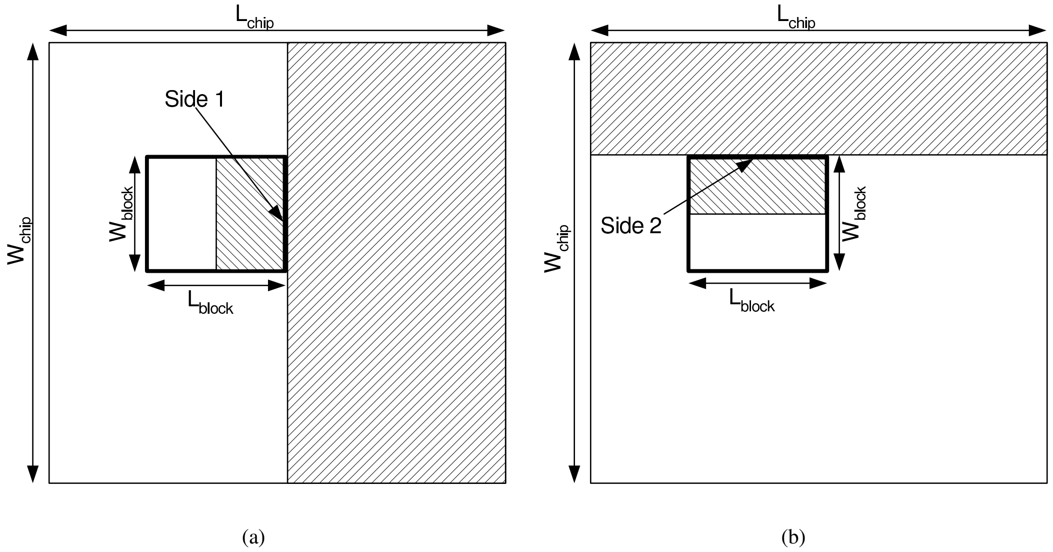


Figure 5. Areas used in calculating lateral resistances.

block. For example, in Figure 5a, Side 1's constriction resistance is the one from the shaded right part of the chip into the right shaded half of that block. For the same block for which we are calculating resistance, now shown in Figure 5b, Side 2's constriction resistance is the one from the shaded top part of the chip into the top shaded part of that block.

The spreading-resistance calculation for the heat spreader and heat sink is shown in Figure 6. The top surface of the heat spreader (heat sink) is divided into five parts. The center is the part that is covered by the chip. Each peripheral, trapezoidal-shaped part of the heat spreader is approximated by two rectangles shown as the shaded areas in figure 4. The two spreading resistances in the figure for the whole model are: 1) from the area covered by the chip to the first rectangle, 2) from the first rectangle to the second rectangle. Fortunately, all these calculations are entirely automated within HotSpot.

This HotSpot model has several advantages over prior models. In contrast to TEMPEST [15], which models the average chip-wide temperature, HotSpot models heat at a much finer granularity and accounts for hotspots in different functional units. HotSpot has more in common with models from our earlier work [41, 42], but with some important improvements. Our first work [41], omitted the lateral resistances and the package, both essential for accurate temperatures and time evolution; and used a rather extreme die thickness of 0.1mm, compressing the spatial temperature gradients too much. Our next model [42] corrected these effects but collapsed the spreader and sink, and did not account for spreading resistance, leading to inconvenient empirical fitting factors and a non-intuitive matching between the physical structure and the model.

Normally, the package-to-air resistance ($R_{convection}$) would be calculated from the specific heat-sink configuration. In

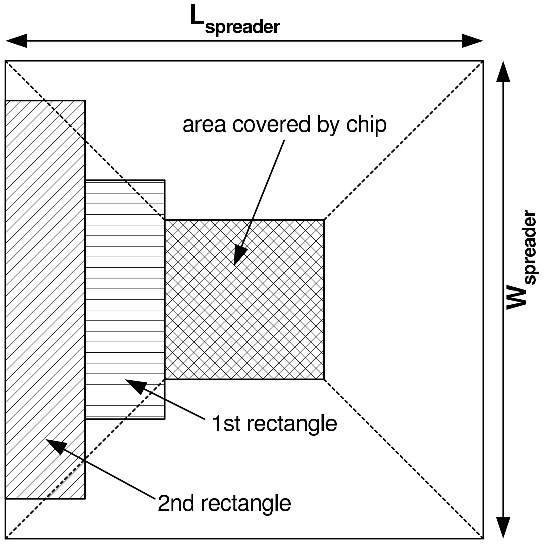


Figure 6. Calculating spreading resistance for the heat spreader or sink.

this paper, we instead manually choose a resistance of 0.8 K/W that gives us a good distribution of benchmark behaviors; see Section 5.5.

HotSpot does still require scaling factors for the capacitances. Capacitors in the spreader and sink must be multiplied by about 0.4. This is a consequence of using a simple single-lump C for each block, rather than a distributed model. The scaling factor we determine empirically matches the value predicted by [30]. Capacitors in silicon, on the other hand, must be multiplied by about 2.2[1] for a 16mm $\times$ 16mm chip of thickness 0.5 mm, and as much as 9.3 for a thickness of 0.1 mm. The reason for this is that, for transient purposes, the bottom surface of the chip is not actually isothermal, whereas HotSpot treats the bottom surface as such by feeding all vertical Rs for the die into a single node. The true isothermal surface lies somewhere in the heat spreader, which means that the "effective thermal mass" that determines the transient behavior of on-die heating is larger than the die thickness and makes the effective thermal capacitance larger than what Equation 3 yields. The scaling factor therefore computes the effective thermal mass. Since the conference paper, we have derived an analytic expression for this scaling factor of capacitances in silicon, whereas before it was merely an empirically determined fitting factor. Note that in effect we are distributing the capacitance for the center of the spreader across the capacitances of the various silicon blocks. It may seem that the capacitance of the spreader's center block should therefore be reduced or removed, but it also plays a role in lateral heat flow. In practice, we have found that the presence or absence of this particular capacitor in the center of the spreader has no perceptible effect on the temperature model once the scaling factor for silicon is in place, so we simply leave it in place, un-modified.

Specifically, following the treatment in [6], we can derive the following expression for this silicon scaling factor α to account for the greater thermal mass as a function of chip area, chip thickness, and heat-spreader thickness. For a silicon chip with length and width both L, and for the spreader with thickness σ, the effective heat-conducting surface area is a function of the heat-spreader's thickness and is given by $(L + 0.88\sigma)^2$. This statement is valid as long as L/σ is greater than 0.4. In our current model, $L/\sigma = 10/1 = 10 \gg 0.4$. We also make the assumption (verified by our Floworks simulations) that the bottom surface of the spreader is indeed an isothermal surface. So the effective volume in the copper spreader that is enclosed by the conducting surface can be approximated by $(L + 0.88\sigma)^2 \cdot \sigma$. This volume plus the silicon chip volume is accounted for in the total transient heat response of the chip. In order to find α, the volume term $(L + 0.88\sigma)^2$ for the copper spreader must be transformed into an equivalent volume of silicon with the same thermal mass, that is $(3.55/1.75) \cdot [(L + 0.88\sigma)^2 \cdot \sigma]$. Here $3.55 \times 10^6 J/m^3 \cdot K$ is thermal mass per volume for copper, and 1.75×10^6 is for silicon. So

$$\alpha = 0.4 \cdot \left(\left[\frac{\frac{3.55}{1.75} \cdot (L + 0.88\sigma)^2 \cdot \sigma}{L \cdot L \cdot t} \right] + 1 \right) \tag{4}$$

Here $L \cdot L \cdot t$ is the volume of the silicon chip. The "+1" term counts for the silicon chip volume itself. And 0.4 is the factor

[1]This value is a refinement of the approximate 2.0 value given in the conference paper.

t	α
0.1	9.3
0.2	4.8
0.3	3.3
0.4	2.6
0.5	2.2
0.6	1.9
0.7	1.6
0.8	1.5
0.9	1.4
1.0	1.3

Table 2. Range of values for α for different chip thicknesses. The value of α is not sensitive to spreader thickness as long as the chip's lateral dimensions are much larger than the spreader thickness. These figures are for a 16mm $\times$ 16mm chip.

for using a lumped model instead of a distributed model. For example, in our Floworks model, $L = 16$mm for the silicon chip, $\sigma = 1$mm for the copper spreader, and the chip thickness $t = 0.5$mm. So we can find $\alpha = 2.2$. It is important to note that, over the range of interesting chip thicknesses, the value of α is quite sensitive to chip thickness t, with a range of values shown in Table 2. On the other hand, α is not sensitive to power densities or the range of power densities for reasonable values that could be observed on a realistic chip.

We could avoid this scaling factor by modeling separate blocks in the spreader that correspond to each block in the silicon. But this would approximately double the size of the matrix for solving the RC circuit. Although having all the vertical resistances for silicon meet at a common node corresponding to just one block in the center of the spreader is indeed an approximation, we feel that it is a reasonable one because there is fairly little heat flow laterally in the spreader: at steady state, the bottom (not the top) of the chip—the surface against the spreader—is close to isothermal, with temperature variations across the bottom surface of less than 10%. On the other hand, vertical heat flow from the chip into the spreader and then into the sink is larger than the lateral heat flow between silicon blocks. In short, this means that the main purpose of the scaling factor for silicon capacitances is to account for the fact that when modeling transient behavior, any change in heat dissipation causes heat flow between the chip and the spreader, so the thermal mass that is affected by short-term heat flows includes the center portion of the spreader. Lateral thermal diffusion in the silicon remains important, as seen by our validation results below and our results for the "migrating computation" technique (see Section 6).

To convey some sense of the R and C values that our model produces, Table 3 gives some sample values.[2] Vertical Rs for the copper spreader and sink correspond to the middle block, while lateral Rs for the spreader and sink correspond to the inner R for one of the peripheral blocks. As mentioned, the convection resistance of 0.8 K/W has been chosen to provide a useful range of benchmark behaviors, and represents a midpoint in the range 0.1-2.0 that might be found for typical heat sinks [48] in desktop and server computers. As less expensive heat sinks are chosen—for use with DTM, for example—the resistance increases.

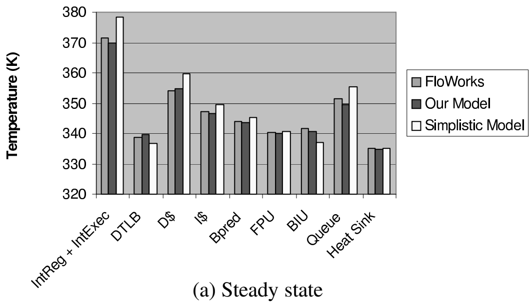
Block	R_{vert}	R_{lat}	C
IntReg+IntExec	0.420	3.75	0.024
D-cache	0.250	6.50	0.032
Spreader (center)	0.025	0.83	0.350
Sink (center)	0.026	0.50	8.800
Convec.	0.800	na	140.449

Table 3. Sample R and C values. Rs are in K/W, and Cs in J/K and include the appropriate scaling factor.

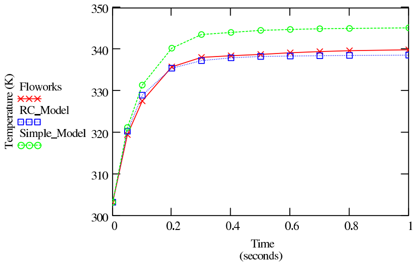
[2] Note that the value of C for the D-cache was erroneously listed in the conference paper as 0.137.

HotSpot dynamically generates the RC circuit when initialized with a configuration that consists of the blocks' layout and their areas (see Section 3.5). The model is then used in a dynamic architectural power/performance simulation by providing HotSpot with dynamic values for power density in each block (these are the values for the current sources) and the present temperature of each block. We use power densities obtained from Wattch, averaged over the last 10K clock cycles; see Section 5.1 for more information on the choice of sampling rate. At each time step, the differential equations describing the RC circuit are solved using a fourth-order Runge-Kutta method (with an adaptive number of iterations depending on the calling interval), returning the new temperature of each block. Each call to the solver takes about $50\mu s$ on a 1.6GHz Athlon processor, so the overhead on simulation time is negligible for reasonable sampling rates, usually less than 1% of total simulation time.

A final note regards the temperature-dependence of conductivity. This temperature dependence only affects results for relatively large temperature ranges greater than 50° [31]. Our current chip model has a much smaller range in active mode, so we use the thermal conductivity of silicon at 85°, our specified maximum junction temperature. Incorporating a temperature dependence will be a necessary extension to HotSpot for working with larger temperature ranges.

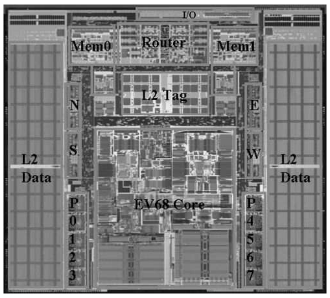


(a) Steady state

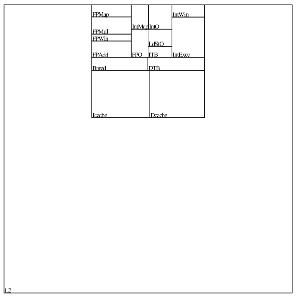


(b) Transient

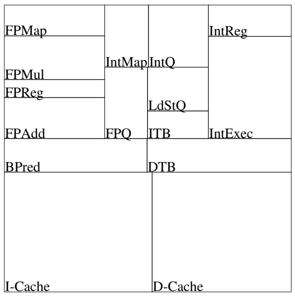
Figure 7. Model validation. (a): Comparison of steady-state temperatures between Floworks, HotSpot, and a "simplistic" model with no lateral resistances. (b): Comparison of the step response in Floworks, HotSpot, and the simplistic model for a single block, the integer register file.



(a) 21364 die photo



(b) Equiv. floorplan



(c) CPU core

Figure 8. (a): Die photo of the Compaq Alpha 21364 [14]. (b): Floorplan corresponding to the 21364 that is used in our experiments. (c): Closeup of 21364 core.

3.4. Calibrating the Model

Dynamic compact models are well established in the thermal-engineering community, although we are unaware of any that have been developed to describe transient heat flow at the granularity of microarchitectural units. Of course, the exact compact model must be validated to minimize errors that might arise from modeling heat transfer using lumped elements. This is difficult, because we are not aware of any source of localized, time-dependent measurements of physical temperatures that could be used to validate our model. It remains an open problem in the thermal-engineering community how to obtain such direct measurements. Eventually, we hope to use thermal test chips (*e.g.*, [5]) or a system like an infrared camera or IBM's PICA picosecond imaging tool [29] to image heat dissipation on a fine temporal and spatial granularity. Until this becomes feasible, we are using a thermodynamics and computational fluid-dynamics tool as a semi-independent model which we can use for validation and calibration.

Our best source of reference is currently a finite-element model in Floworks (http://www.floworks.com), a commercial, finite-element simulator of 3D fluid and heat flow for arbitrary geometries, materials, and boundary conditions. We model a 0.5mm-thick (of which the top 0.05mm of is actually generating heat), 1cm $\times$ 1cm silicon die with various floorplans and various power densities in each block; this die is attached to a 1mm-thick, 3cm $\times$ 3cm copper spreader and a 6cm $\times$ 6cm copper heat sink on one side, and covered by a 1mm-thick, 3cm $\times$ 3cm insulating cap (currently an ideal, insulating material) on the other side. The heat sink has 8 fins, each 20.6cm high on a 6.9mm-thick base; and the cap has a notch on its interior surface to accommodate the die. Air is assumed to stay fixed at 45°C; airflow is laminar at 10 m/s. In Floworks, volume heat generation rate is assigned to each silicon surface block according to the power numbers acquired from Wattch. Note that to simplify the geometry, the integer register file and integer execution units have been combined, with the area-weighted average power density used in the Floworks simulation. All materials' properties are defined in the Engineering Database of Floworks. Figure 9 shows a view of our Floworks model, with the insulating cap removed to show the die.

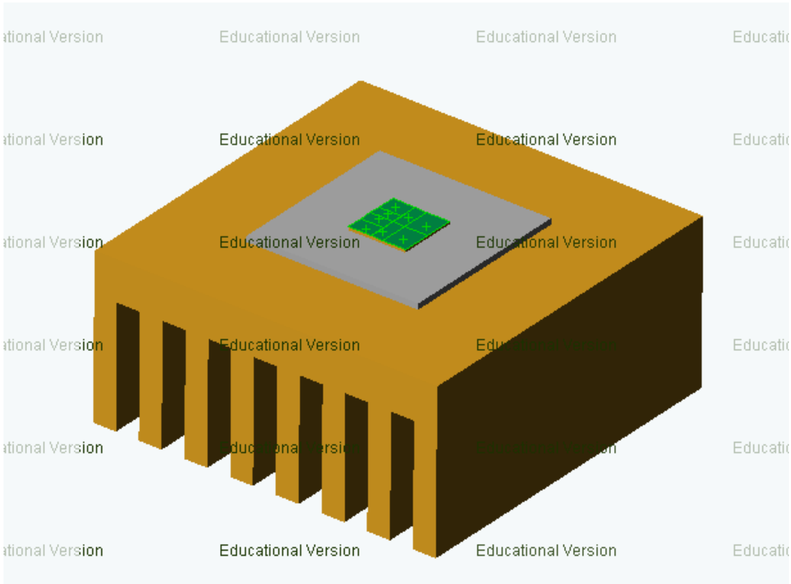


Figure 9. Our Floworks model, with the insulating cap removed to show the die.

Floworks and HotSpot are not entirely independent. Although all R, C, and scaling factors are determined analytically, without reference to Floworks results (an improvement over what is claimed in our conference paper), Floworks simulations did help to guide our development of the HotSpot model. In any case, we can verify that the two obtain similar steady-state operating temperatures and transient response. Figure 7a shows steady state validation comparing temperatures predicted by Floworks, HotSpot, and a "simplistic" model that eliminates the lateral portion of the RC circuit (but not the package, omission of which would yield extremely large errors). HotSpot shows good agreement with Floworks, with errors (with respect to the ambient, 45°C or 318°K) always less than 5.8% and usually less than 3%. The simplistic model, on the other hand, has larger errors, as high as 16%. One of the largest errors is for the hottest block, which means too many thermal triggers will be generated. Figure 7b shows transient validation comparing, for Floworks, HotSpot, and the simplistic model, the evolution of temperature in one block on the chip over time. The agreement is excellent between Floworks and HotSpot, but the simplistic model shows temperature rising too fast and too far. Both the steady-state and the transient results show the importance of thermal diffusion in determining on-chip temperatures.

We have also validated the scaling factor for the silicon capacitances by testing a 0.1mm-thick chip with a copper spreader of the same size, and our baseline 0.5mm chip but with an aluminum ($2.58 \times 10^6 J/m^3 \cdot K$) spreader of the same size.

3.5. Floorplanning: Modeling Thermal Adjacency

The size and adjacency of blocks is a critical parameter for deriving the RC model. In all of our simulations so far, we have used a floorplan (and also approximate microarchitecture and power model) corresponding to that of the Alpha 21364. This floorplan is shown in Figure 8. Like the 21364, it places the CPU core at the center of one edge of the die, with the surrounding area consisting of L2 cache, multiprocessor-interface logic, etc. Since we model no multiprocessor workloads, we omit the multiprocessor interface logic and treat the entire periphery of the die as second-level (L2) cache. The area of this cache seems disproportionately large compared to the 21364 die photo in Figure 8a, because we have scaled the CPU to 130nm while keeping the overall die size constant. Note that we do not model I/O pads, because we do not yet have a good dynamic model for their power density—more future work.

When we vary the microarchitecture, we currently obtain the areas for any new blocks by taking the areas of 21264 units and scaling as necessary. When scaling cache-like structures, we use CACTI 3.0 [39], which uses analytic models to derive area. Since CACTI's a-priori predictions vary somewhat from the areas observed in the 21264, we use known areas as a starting point and only use CACTI to obtain scaling factors.

Eventually, we envision an automated floorplanning algorithm that can derive areas and floorplans automatically using only the microarchitecture configuration.

3.6. Limitations

There clearly remain many ways in which the model can be refined to further improve its accuracy and flexibility, all interesting areas for future research. Many of these require advances in architectural power modeling in addition to thermal modeling, and a few—like modeling a wider range of thermal packages and including the effects of I/O pads—were mentioned above. We reiterate that our approach is motivated by an microarchitecture-centric viewpoint. This means that we neglect a number of issues that are typically considered in a full thermal design. For example, we completely neglect thermal modeling of the circuit board, heating of the air within the computer system case, temperature non-uniformities in this air, heat flow through the package pins, etc. Because microarchitecture simulations only model very small time periods of perhaps a few seconds at most, these components that are external to the chip and its package respond too slowly to change temperature during these short time scales, and these external components do not contribute in a significant way to the capacitance being driven by the chip. Indeed, the heat sink itself does not change temperature on these time scales, but it must be included because its capacitance does affect the transient behavior of the chip.

Perhaps the most important and interesting area for future work is the inclusion of heating due to the clock grid and other interconnect. The effects of wires can currently be approximated by including their power dissipation in the dynamic, per-block power-density values that drive HotSpot. A more precise approach would separately treat self-heating in the wire itself and heat transfer to the surrounding silicon, but the most accurate way to model these effects is not clear. Another important consideration is that activity in global wires crossing a block may be unrelated to activity within a block. Wire effects may therefore be important, for example by making a region that is architecturally idle still heat up.

Another issue that requires further study is the appropriate granularity at which to derive the RC model. HotSpot is flexible: blocks can be specified at any desired granularity, but as the granularity increases, the model becomes more complex and more difficult to reason about. We are currently using HotSpot with blocks that correspond to major microarchitectural units, but for units in which the power density is non-uniform—a cache, for example—this approximation may introduce some imprecision. Preliminary results suggest that when the non-uniformity is on a scale smaller than the die's "equivalent thickness" mentioned above—as in the case of individual lines in a cache—the effect on temperature is negligible, but this requires further study.

At the packaging level, we have neglected the interface layers between the die and spreader and the spreader and heat sink. Possible interface materials range from very inexpensive pads with fairly high resistance, to thermal greases and phase-change films that conform better to the mating surfaces. These higher-quality interface materials should exhibit very small thermal resistances that can be combined with the spreader and sink layers, and completely negligible capacitances, but should eventually be included for completeness and to allow the user to explore different materials.

All these effects should be included in the model while maintaining the direct physical correspondence between the model and the structure being modeled, thus preserving the property that it is easy to reason about heat flow and the role of any particular element. This means that the model should ideally have no factors that do not have a physical explanation and derivation, and the model should be parameterized to allow the user to explore different configurations.

Finally, although HotSpot is based on well-known principles of thermodynamics and has been validated with a semi-independent FEM, further validation is needed, preferably using real, physical measurements from a processor running realistic workloads.

3.7. Importance of Directly Modeling Temperature

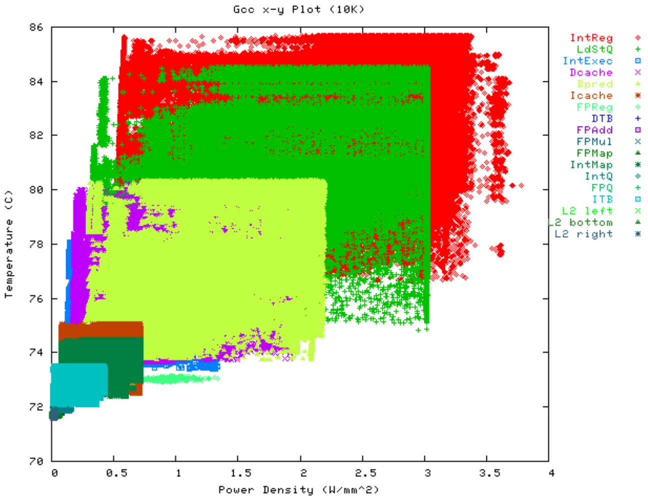
Due to the lack of an architectural temperature model, a few prior studies have attempted to model temperatures by averaging power dissipation over a window of time. This will not capture any localized heating unless it is done at the granularity of on-chip blocks, but even then it fails to account for lateral coupling among blocks, the role of the heat sink, the non-linear rate of heating, etc. We have also encountered the fallacy that temperature corresponds to instantaneous power dissipation, when in fact the thermal capacitance acts as a low-pass filter in translating power variations into temperature variations.

Units	Avg. Temp.	R^2 (%)					
	(°C)	10K	100K	1M	10M	100M	1B
Icache	74.4	43.9	51.1	55.8	73.8	78.5	10.5
ITB	73.2	35.3	42.2	46.8	64.0	75.0	10.6
Bpred	76.2	54.0	71.5	77.6	88.7	91.0	5.3
IntReg	83.5	44.2	51.9	57.0	76.4	71.0	8.0
IntExec	76.7	46.3	53.3	57.9	75.7	76.6	8.3
IntMap	73.9	41.7	49.6	54.8	73.5	76.8	8.0
IntQ	72.4	31.5	36.4	39.6	53.9	80.7	13.0
LdStQ	79.2	47.9	63.4	69.0	83.6	83.2	6.6
Dcache	77.3	46.8	60.5	65.9	81.2	82.8	10.8
DTB	72.0	29.6	38.2	41.7	53.4	87.5	16.4
FPReg	73.0	26.0	29.6	38.8	64.6	84.8	21.1
FPAdd	72.6	49.7	51.1	54.9	66.5	86.4	24.9
FPMul	72.6	53.9	54.1	54.9	62.1	84.8	29.6
FPMap	71.7	16.8	20.2	22.3	26.9	0.5	3.2
FPQ	71.8	28.0	30.0	35.2	49.4	78.0	30.7
L2	71.7	14.2	19.7	21.8	26.6	49.9	3.3

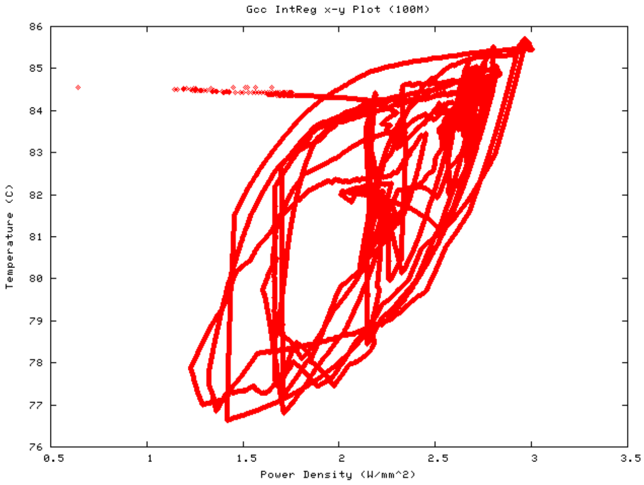
Table 4. Correlation of average power vs. temperature for power averaging windows of 10K–1B cycles.

To show the importance of using a thermal model instead of a power metric, we have computed the R^2 value for correlation between temperature and a moving average of power dissipation for different averaging intervals. The R-squared value gives the percentage of variance that is common between two sets of data; values closer to 100% indicate better correlation. The data in Table 4 come from *gcc*, a representative benchmark, with the reference *expr* input. Temperatures were collected using HotSpot, and power measurements were collected using various averaging periods, simulating *gcc* to completion (about 6 billion cycles). Average temperatures differ slightly from those reported in the conference paper due to the longer simulation interval.

The best averaging interval is 100 million cycles. But the high R^2 values are misleading, because even though statistical *correlation* is reasonably high for this averaging window, power still cannot be used to effectively infer operating *temperature* with any useful precision. Figures 10a and 10b present, for two different averaging intervals, scatter plots for each value of average power density vs. the corresponding temperature. Although the 100 M interval does show correlation, large temperature ranges (y-axis) are observed for any given value of power density (x-axis). This is partly due to the exponential nature of heating and cooling, which can be observed in the exponentially rising and falling curves.



(a) 10 K



(b) 100 M, I-reg only

Figure 10. Scatter plots of temperature vs. average power density for gcc with averaging intervals of (a) 10K, showing all the major blocks, and (b) 100M cycles, showing only the integer register file.

4. Techniques for Architectural DTM

This section describes the various architectural mechanisms for dynamic thermal management that are evaluated in this paper, including both extant techniques and those that we introduce, and discusses how sensor imprecision affects thermal management. It is convenient to define several terms: the *emergency threshold* is the temperature above which the chip is in *thermal violation*; for $85°$, violation may result in timing errors, while for lower-performance chips with higher emergency thresholds, violation results in higher error rates and reduced operating lifetime. In either case, we assume that the chip should *never* violate the emergency threshold. This is probably overly strict, since error rates and aging are probabilistic phenomena, and sufficiently brief violations may be harmless, but no good architecture-level models yet exist for a more nuanced treatment of these thresholds. Finally, the *trigger threshold* is the temperature above which runtime thermal management begins to operate; obviously, trigger $<$ emergency.

4.1. Runtime Mechanisms

This paper proposes three new architecture techniques for DTM: "temperature-tracking" frequency scaling, local toggling, and migrating computation. They are evaluated in conjunction with four techniques that have previously been proposed, namely DVS (but unlike prior work, we add feedback control), global clock gating (where we also add feedback control), feedback-controlled fetch toggling, and a low-power secondary pipeline. Each of the techniques is described below.

For techniques which offer multiple possible settings, we use formal feedback control to choose the setting. Feedback control allows the design of simple but robust controllers that adapt behavior to changing conditions. Following [47], we use PI (proportional-integral) controllers, comparing the hottest observed temperature during each sample against the setpoint. The difference e is multiplied by the gain K_c to determine by how much the controller output u should change, *i.e.*:

$$u[k] = u[k-1] + K_c \cdot e[k-1] \qquad (5)$$

This output is then translated proportionally into a setting for the mechanism being controlled. The hardware to implement this controller is minimal. A few registers, an adder, and a multiplier are needed, along with a state machine to drive them. But single-cycle response is not needed, so the controller can be made with minimum-sized circuitry. The datapath width in this circuit can also be fairly narrow, since only limited precision is needed.

As mentioned earlier, Brooks and Martonosi [8] pointed out that for fast DTM response, interrupts are too costly. We adopt their suggestion of on-chip circuitry that directly translates any signal of thermal stress into actuating the thermal response. We assume that it simply consists of a comparator for each digitized sensor reading, and if the comparator finds that the temperature exceeds the trigger, it asserts a signal. If any trigger signal is asserted, the appropriate DTM technique is engaged.

Next we describe the new techniques introduced in this paper, followed by the other techniques we evaluate.

Temperature-Tracking Frequency Scaling. Dynamic voltage scaling (DVS) is typically preferred for power and energy conservation over dynamic frequency scaling (DFS), because DVS gives cubic savings in power density relative to frequency. However, *independently* of the relationship between frequency and voltage, the temperature-dependence of carrier mobility means that frequency is also linearly dependent on the operating *temperature*. Garrett and Stan [17] report an 18% variation over the range 0-100°.

This suggests that the standard practice of designing the nominal operating frequency for the maximum allowed operating temperature is too conservative. When applications exceed the temperature specification, they can simply scale frequency down in response to the rising temperature. Because this temperature dependence is mild within the interesting operating region, the performance penalty of doing so is also mild—indeed, negligible.

For each change in setting, DVS schemes must stall for anywhere from 10–50 μs to accommodate resynchronization of the clock's phase-locked loop (PLL), but if the transition is gradual enough, the processor can execute through the change without stalling, as the Xscale is believed to do [37].

We examine a discretized frequency scaling with 10 MHz steps and 10μs stall time for every change in the operating frequency; and an ideal version that does not incur this stall but where the change in frequency does not take effect until after 10μs has elapsed. We call these "TT-DFS" and "TT-DFS-i(deal)". Larger step sizes do not offer enough opportunity to adapt, and smaller step sizes create too much adaptation and invoke too many stalls.

This technique is unique among our other techniques in that the operating temperature may legitimately exceed the 85° threshold that other techniques must maintain. As long as frequency is adjusted before temperature rises to the level where timing errors might occur, there is no violation.

No feedback control is needed for TT-DFS, since the frequency is simply a linear function of the current operating temperature. It might seem odd, given the statement that DFS is inferior to DVS, that we only scale frequency. The reason is that the dependence of frequency on temperature is independent of its dependence on voltage: any change in voltage requires an additional reduction in frequency. This means that, unlike traditional DFS, TT-DFS does not allow reductions in voltage without further reductions in frequency.

Local Feedback-Controlled Fetch Toggling. A natural extension of the feedback-controlled fetch toggling proposed in [41] is to toggle individual domains of the processor at the gentlest duty cycle that successfully regulates temperature: "PI-LTOG". Only units in thermal stress are toggled. By toggling a unit like the integer-execution engine at some duty cycle of x/y, we mean that the unit operates at full capacity for x cycles and then stalls for $y - x$ cycles. The choice of duty cycle is a feedback-control problem for which we use the PI controller with a gain of 1 (except the integer domain which uses a gain of 3) and a setpoint of 81.8°.

In our scheme, we break the processors into the following domains, each of which can be independently toggled:

- *Fetch engine*: I-cache, I-TLB, branch prediction, and decode.
- *Integer engine*: Issue queue, register file, and execution units.
- *FP engine*: Issue queue, register file, and execution units.
- *Load-store engine*: Load-store ordering queue, D-cache, D-TLB, and L2-cache.

Note that decoupling buffers between the domains, like the issue queues, will still dissipate some power even when toggled off in order to allow neighboring domains to continue operating; for example, allowing the data cache to write back results even though the integer engine is stalled that cycle.

Depending on the nature of the workload's ILP and the degree of toggling, localization may reduce the performance penalties associated with toggling or GCG, but when the hot unit is also on the critical execution path, toggling that unit off will tend to slow the entire processor by a corresponding amount.

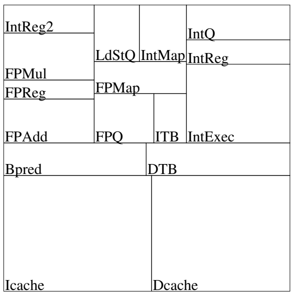


Figure 11. Floorplan with spare integer register file for migrating computation.

Migrating Computation. Two units that run hot by themselves will tend to run even hotter when adjacent. On the other hand, separating them will introduce additional communication latency that is incurred regardless of operating temperature. This suggests the use of spare units located in cold areas of the chip, to which computation can *migrate* only when the primary units overheat.

We developed a new floorplan that includes an extra copy of the integer register file, as shown in Figure 11. When the primary register file reaches $81.6°$, issue is stalled, instructions ready to write back are allowed to complete, and the register file is copied, four values at a time. Then all integer instructions use the secondary register file, allowing the primary register file to cool down while computation continues unhindered except for the extra computational latency incurred by the greater communication distance. The extra distance is accounted for by charging one extra cycle[3] for every register-file access. (For simplicity in our simulator, we approximate this by simply increasing the latency of every functional unit by one cycle, even though this yields pessimistic results.) When the primary register file returns to $81.5°$, the process is reversed and computation resumes using the primary register file. We call this scheme "MC". Note that, because there is no way to guarantee that MC will prevent thermal violations, a failsafe mechanism is needed, for which we use PI-LTOG.

It is also important to note that the different floorplan will have some direct impact on thermal behavior even without the use of any DTM technique. The entire integer engine runs hot, and even if the spare register file is never used, the MC floorplan spreads out the hot units, especially by moving the load-store queue (typically the second- or third-hottest block) farther away.

Another important factor to point out is that driving the signals over the longer distance to the secondary register file will require extra power that we currently do not account for, something that may reduce MC's effectiveness, especially if the drivers are close to another hot area.

The design space here is very rich, but we were limited in the number of floorplans that we could explore, because developing new floorplans that fit in a rectangle without adding whitespace is a laborious process. Eventually, we envision an automated floorplanning algorithm that can derive floorplans automatically using some simple specification format.

The dual-pipeline scheme proposed by Lim *et al.* [26] could actually be considered another example of migrating computation. Because the secondary, scalar pipeline was designed mainly for energy efficiency rather than performance, the dual-pipeline scheme incurred the largest slowdowns of any scheme we studied, and we compare this separately to our other schemes. MC could also be considered a limited form of multi-clustered architecture [11].

Dynamic Voltage Scaling. DVS has long been regarded as a solution for reducing energy consumption, has recently been proposed as one solution for thermal management [8, 20], and is used for this purpose in Transmeta's Crusoe processors [16]. The frequency must be reduced in conjunction with voltage since circuits switch more slowly as the operating voltage approaches the threshold voltage. This reduction in frequency slows execution time, especially for CPU-bound applications, but DVS provides a cubic reduction in power density relative to frequency.

We model two scenarios that we feel represent the range of what will likely be available in the near future. In the first ("PI-DVS"), there are ten possible discrete DVS settings ranging from 100% of the nominal voltage to 50% in equal steps. The penalty to change the DVS setting is 10μs, during which the pipeline is stalled. In the second ("PI-DVS-i(deal)"), the processor may continue to execute through the change but the change does not take effect until after $10\mu s$ have elapsed, just as with TT-DFS-i.

[3] In our conference paper, this was incorrectly stated as two cycles.

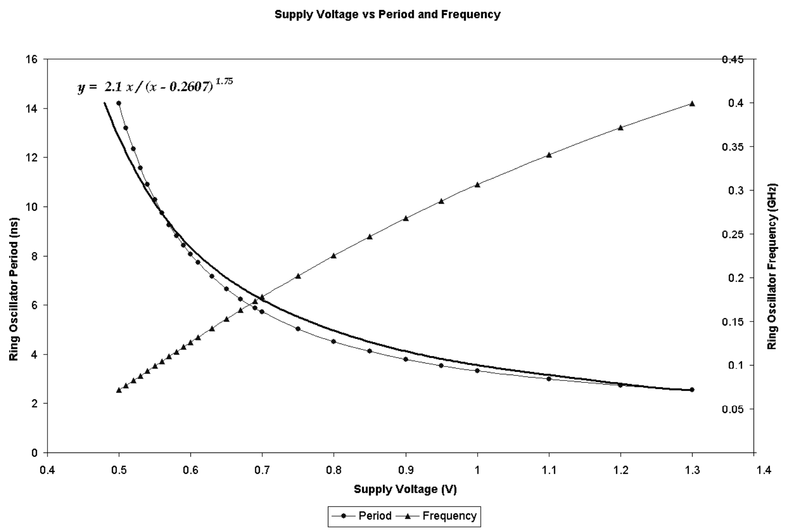


Figure 12. Simulated and calculated operating frequency for various values of V_{dd}. The nominal operating point of our simulated processor is 3 GHz at 1.3V.

Because the relationship between voltage and frequency is not linear but rather is given by [30]

$$\frac{k(V_{dd} - V_t)^a}{V_{dd}} \tag{6}$$

voltage reductions below about 25% of the nominal value will start to yield disproportionate reductions in frequency and hence performance. We used Cadence with BSIM 100nm low-leakage models to simulate the period of a 101-stage ring oscillator under various voltages to determine the frequency for each voltage step (see Figure 12). Fitting this to a curve, we determined that $k = 2.1$ and $a = 1.75$, which matches values reported elsewhere, *e.g.*, [22]. The appropriate values were then placed in a lookup table in the simulator. For continuous DVS, we perform linear interpolation between the table entries to find the frequency for our chosen voltage setting.

To set the voltage, we use a PI controller with a gain of 10 and a setpoint of $81.8°$. A problem arises when the controller is near a boundary between DVS settings, because small fluctuations in temperature can produce too many changes in setting and a $10\mu s$ cost each time that the controller does not take into account. To prevent this, we apply a low-pass filter to the controller output when voltage is to be scaled up. The filter compares the performance cost of the voltage change to the performance benefit of increasing the voltage and frequency and makes the change only when profitable. Both these cost and benefit measures are percentages of change in delay. Computation of the benefit is straightforward: current delay is just the reciprocal of the current clock frequency. Similarly, future delay is also computed and the percent change is obtained from these numbers. However, in order to compute the cost, knowledge of how long the program will run before incurring another voltage switch is necessary, because the switch time is amortized across that duration. We take a simple prediction approach here, assuming the past duration to be indicative of the future and using it in the computation instead of the future duration. So, the ratio of the switch time and the past duration gives the cost. Note that this filter cannot be used when the voltage is to be scaled down because scaling down is mandatory to prevent thermal emergency.

Global Clock Gating and Fetch Toggling. As a baseline, we consider global clock gating ("GCG") similar to what the Pentium 4 employs [18], in which the clock is gated when the temperature exceeds the trigger of $81.8°$ and ungated when the temperature falls back below that threshold. We also consider a version in which the duty cycle on the clock gating is determined by a PI controller with a gain of 1 ("PI-GCG"), similar to the way PI-LTOG is controlled. We recognize that gating the entire chip's clock at fine duty cycles may cause voltage-stability problems, but it is moot for this experiment. We only seek to determine whether PI-GCG can outperform PI-LTOG, and find that it cannot because it slows down the entire chip while PI-LTOG exploits ILP.

We also evaluated fetch toggling [8], and a feedback controlled version in which fetching rather than the clock is gated until the temperature reaches an adequate level. Overall, fetch toggling and global clock gating are quite similar. We model global clock gating because it also cuts power in the clock tree and has immediate effect.

While the clock signal is gated, power dissipation within the chip is eliminated except for leakage power. Global clock gating is therefore a "duty-cycle based technique" for approximating traditional DFS, but without any latency to change the "frequency". The Pentium 4 uses a duty cycle of 1/2, where the clock is enabled for 2μs and disabled for 2μs, and once triggered, the temperature must drop below the trigger threshold by one degree before normal operation resumes [21]. In the Pentium 4, each change in the clock frequency requires a high-priority interrupt, which Gunther *et al.* [18] report takes approximately 1μs (but Lim *et al.* [26] report 1 ms). Brooks and Martonosi [8] instead proposed fetch *toggling*, in which fetch is simply halted until the temperature reaches an adequate level—they called this "toggle1." This has two minor drawbacks compared to clock gating, in that power dissipation takes a few cycles to drop (as the pipeline drains) and the power dissipation in the clock tree (15% or more [28]) is not reduced. Brooks and Martonosi also considered setting the duty cycle on fetch to 1/2 ("toggle2"), but they and also Skadron *et al.* [41] found that this did not always prevent thermal violations. We believe the reason that the P4 succeeds with a duty cycle of 1/2 is that each phase is so long—microseconds rather than nanoseconds—that the chip can cool down sufficiently well. On the other hand, the penalty can be excessive when only minimal cooling is required.

Overall, fetch toggling and global clock gating are quite similar. We model global clock gating ("GCG") separately because it also cuts power in the clock tree and has immediate effect. For the feedback-controlled versions of both schemes, we use a gain of 1.

Low-Power Secondary Pipeline. Lim, Daasch, and Cai [26] proposed, instead of migrating accesses to individual units, to use a secondary pipeline with very low-power dissipation. We refer to this technique as "2pipe." Whenever the superscalar core overheats anywhere, the pipeline is drained, and then an alternate scalar pipeline is engaged. This pipeline shares the fetch engine, register file, and execution units of the superscalar pipeline; because they are now accessed with at most one instruction per cycle, their power dissipation will fall, but it is only the out-of-order structures whose active power dissipation is completely reduced. This scheme is essentially an aggressive version of computation migration, but we find that it penalizes performance more than necessary.

In [26], they do not model the extra latency that may be associated with accessing the now-disparate units, so we neglect this factor as well, even though we account for such latency in our "MC" technique. We also make the optimistic assumption here that when the low-power secondary pipeline is engaged, zero power is dissipated in the out-of-order units after they drain. We charge 1/4 the power dissipation to the integer-execution unit to account for the single-issue activity. These idealized assumptions are acceptable because they favor this scheme, and we still conclude that it is inferior to simpler alternatives like our floorplan-based techniques or even DVS alone. Of course, it is important to repeat that this technique was not optimized for thermal management but rather for energy efficiency.

4.2. Sensors

Runtime thermal management requires real-time temperature sensing. So far, all prior published work of which we are aware has assumed omniscient sensors, which we show in Section 6 can produce overly optimistic results. Sensors that can be used on chip for the type of localized thermal response we contemplate are typically based on analog CMOS circuits using a current reference. An excellent reference is [2]. The output current is digitized using a ring oscillator or some other type of delay element to produce a square wave that can be fed to a counter. Although these circuits produce nicely linear output across the temperature range of interest, and respond rapidly to changes in temperature, they unfortunately are sensitive to lithographic variations and supply-current variations. These sources of imprecision can be reduced by making the sensor circuit larger, at the cost of increased area and power. Another constraint that is not easily solved by up-sizing is that of sensor bandwidth—the maximum sampling rate of the sensor.

Our industry contacts tell us that CMOS sensors which would be reasonable to use in moderate quantity of say 10–20 sensors would have at best a precision of $\pm 2°$C and sampling rate of 10 microseconds. This matches the results in [2]. We place one sensor per architectural block.

We model the imprecision by randomizing at each node the true temperature reading over the specified range $\pm 2°$. We assume that the hardware reduces the sensor noises at runtime by using a moving average of the last ten measurements, because averaging reduces the error as the square root of the number of samples. This of course assumes that the measured value is stationary, which is not true for any meaningful averaging window. This means we must also account for the potential

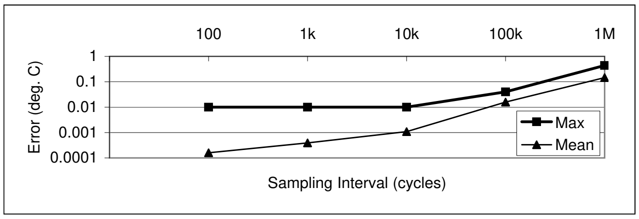


Figure 13. Plot of absolute error in temperature as a function of sampling rate.

change in temperature over the averaging window, which we estimate to be potentially as much as $0.4°$ if temperatures can rise $0.1°$ per $30\mu s$. For $\pm 2°$, we are therefore able to reduce the uncertainty to $S = \frac{2}{\sqrt{10}} + 0.4 = \pm 1°$. An averaging window of ten samples was chosen because the improved error reduction with a larger window is offset by the larger change in the underlying value.

There is one additional non-ideality that must be accounted for when modeling sensors and cannot be reduced by averaging. If a sensor cannot be located exactly coincident with every possible hotspot, the temperature observed by the sensor may be cooler by some spatial-gradient factor G than at the hotspot. If, in addition to the random error discussed above, there is also a systematic or offset error in the sensor that cannot be canceled, this increases the magnitude of the fixed error G. Based on simulations in our finite-element model and the assumption that sensors can be located near but not exactly coincident with hotspots, we choose $G = 2°$.

It can therefore be seen that for any runtime thermal-management technique, the use of sensors lowers the emergency threshold by $G + S$ ($3°$ in our case). This must be considered when comparing to other low-power design techniques or more aggressive and costly packaging choices. It is also strong motivation for finding temperature-sensing techniques that avoid this overhead, perhaps based on clever data fusion among sensors, or the combination of sensors and performance counters.

5. Simulation Setup

In this section, we describe the various aspects of our simulation framework and how they are used to monitor runtime temperatures for the SPEC2000 benchmarks [44].

5.1. Integrating the Thermal Model

HotSpot is completely independent of the choice of power/performance simulator. Adding HotSpot to a power/performance model merely consists of two steps. First, initialization information must be passed to HotSpot. This consists of an adjacency matrix describing the floorplan (the floorplan used for the experiments in this paper is included in the HotSpot release) and an array giving the initial temperatures for each architectural block. Then at runtime, the power dissipated in each block is averaged over a user-specified interval and passed to HotSpot's RC solver, which returns the newly computed temperatures. A time step must also be passed to indicate the length of the interval over which power data is averaged.

Although it is feasible to recompute temperatures every cycle, this is wasteful, since even at the fine granularity of architectural units, temperatures take at least 100K cycles to rise by $0.1°$C. We chose a sampling rate of 10K cycles as the best tradeoff between precision and overhead. For sampling intervals of 10K and less, the error is less than $0.01°$—the same magnitude as the rounding error due to significant digits in the Runge-Kutta solver. Figure 13 plots the maximum and mean error for all the SPEC2k benchmarks as a function of sampling rate. "Max" is the largest absolute error observed for any benchmark. "Mean" considers the mean absolute error for each block, and the largest of these values for any benchmark is plotted. Note that this error rate is determined with respect to a sampling interval of 10 cycles.

The number of iterations for the Runge-Kutta solver is adaptive, to account for the different number of iterations required for convergence at different sampling intervals. Specifically, the step size in the solver is kept constant and the number of iterations is a linear function of the sampling interval. The step size is determined by the product of the required degree of precision in temperature and the maximum RC time constant of the functional units. Since the RC time constant is the reciprocal of the maximum rate at which temperature can rise in the chip, this step size is in fact very conservative. This is an improvement over what was reported in the conference paper, where we used a fixed number of iterations—four—regardless of sampling interval. Now only one iteration is needed for sampling intervals of 11K or less, improving the performance of the solver. The overhead is also now essentially independent of sampling interval. With the new adaptive iteration count, the extra simulation time for thermal modeling is less than 1%.

5.2. Power-Performance Simulator

We use a power model based on power data for the Alpha 21364 [3]. The 21364 consists of a processor core identical to the 21264, with a large L2 cache and (not modeled) glueless multiprocessor logic added around the periphery. An image of the chip is shown in Figure 8, along with the floorplan schematic that shows the units and adjacencies that HotSpot models. Because we study microarchitectural techniques, we use Wattch version 1.02 [9] to provide a framework for integrating our power data with the underlying SimpleScalar [10] architectural model. Our power data was for 1.6 V at 1 GHz in a 0.18μ process, so we used Wattch's linear scaling to obtain power for 0.13μ, V_{dd}=1.3V, and a clock speed of 3 GHz. These values correspond to the recently-announced operating voltage and clock speed that for the Pentium 4 [33]. We assume a die thickness of 0.5mm. Our spreader and sink are both made of copper. The spreader is 1mm thick and 3cm $\times$ 3cm, and the sink has a base that is 7mm thick and 6cm $\times$ 6cm. Power dissipated in the per-block temperature sensors is not modeled.

The biggest difficulty in using SimpleScalar is that the underlying *sim-outorder* microarchitecture model is no longer terribly representative of contemporary processors, so we augmented it to model an Alpha 21364 as closely as possible. We extended both the microarchitecture and corresponding Wattch power interface; extending the pipeline and breaking the centralized RUU into four-wide integer and two-wide floating-point issue queues, 80-entry integer and floating-point merged physical/architectural register file, and 80-entry active list. First-level caches are 64 KB, 2-way, write-back, with 64B lines and a 2-cycle latency; the second-level is 4 MB, 8-way, with 128B lines and a 12-cycle latency; and main memory has a 225-cycle latency. The branch predictor is similar to the 21364's hybrid predictor, and we improve the performance simulator by updating the fetch model to count only one access (of fetch-width granularity) per cycle. The only features of the 21364 that we do not model are the register-cluster aspect of the integer box, way prediction in the I-cache, and speculative load-use issue with replay traps (which may increase power density in blocks that are already quite hot). The microarchitecture model is summarized in Table 5. Finally, we augmented SimpleScalar/Wattch to account for dynamic frequency and voltage scaling and to report execution time in seconds rather than cycles as the metric of performance.

Processor Core		Other Operating Parameters	
Active List	80 entries	Nominal frequency	3 GHz
Physical registers	80	Nominal V_{dd}	1.3 V
LSQ	64 entries	Ambient air temperature	45°C
Issue width	6 instructions per cycle (4 Int, 2 FP)	Package thermal resistance	0.8 K/W
		Die	0.5mm thick, 15.9mm $\times$ 15.9mm
Functional Units	4 IntALU,1 IntMult/Div, 2 FPALU,1 FPMult/Div, 2 mem ports	Heat spreader	Copper, 1mm thick, 3cm $\times$ 3cm
		Heat sink	Copper, 7mm thick, 6cm $\times$ 6cm
Memory Hierarchy		**Branch Predictor**	
L1 D-cache Size	64 KB, 2-way LRU, 64 B blocks, writeback	Branch predictor	Hybrid PAg/GAg with GAg chooser
L1 I-cache Size	64 KB, 2-way LRU, 64 B blocks both 2-cycle latency	Branch target buffer	2 K-entry, 2-way
L2	Unified, 4 MB, 8-way LRU, 128B blocks, 12-cycle latency, writeback	Return-address-stack	32-entry
Memory	225 cycles (75ns)		
TLB Size	128-entry, fully assoc., 30-cycle miss penalty		

Table 5. Configuration of simulated processor microarchitecture.

5.3. Modeling the Temperature-Dependence of Leakage

Because leakage power is an exponential function of temperature, these power contributions may be large enough to affect the temperature distribution and the effectiveness of different DTM techniques. Furthermore, leakage is present regardless of activity, and leakage at higher temperatures may affect the efficacy of thermal-management techniques that reduce only activity rates. Eventually, we plan to combine HotSpot with our temperature/voltage-aware leakage model [49] to more precisely track dynamic leakage-temperature interactions, which we believe are an interesting area for future work. For now, to make sure that leakage effects are modeled in a reasonable way, we use a simpler model: like Wattch, leakage in each unit is simply treated as a percentage of its power when active, but this percentage is now determined based on temperature and technology node using figures from ITRS data [40].

The original model assumes that idle architectural blocks are clock-gated and leak a fixed percentage of the dynamic power that they would dissipate if active. The default value of 8% happens to correspond to the leakage percentage that would be seen at 85 C in the 130 nm generation, according to our calculations from the ITRS projections. To incorporate leakage effects in a simple way, we retain Wattch's notion that power dissipated by a block during a cycle in which it is idle can be represented as a percentage of that its active power. The only difference is that this percentage should be a function of temperature. To model this dependence, we use ITRS data [40] to derive an exponential distribution for the ratio of leakage power to dynamic power as a function of temperature, and recompute the leakage ratio at every time step. To model this dependence, we derive an exponential distribution for the ratio R_T of leakage power to dynamic power as a function of temperature T:

$$R_T = \frac{R_0}{V_0 T_0^2} e^{\frac{B}{T_0}} \cdot VT^2 \cdot e^{\frac{-B}{T}} \tag{7}$$

where T_0 is the ambient temperature and R_0 is the ratio at T_0 and nominal voltage V_0. B is a process technology constant that depends on the ratio between the threshold voltage and the subthreshold slope. This ratio was computed using the leakage current and saturation drive current numbers from ITRS 2001. Only the $VT^2 \cdot e^{\frac{-B}{T}}$ term varies with temperature and/or operating voltage. This expression is implemented as a function call that replaces the fixed leakage factor in the original Wattch.

It is desirable to eventually model leakage in more detail, to account for structural details and permit studies of the interactions between temperature and leakage-management techniques. The interaction of leakage energy, leakage control, and thermal control is beyond the scope of this paper, but is clearly an interesting area for future work, and since we do not study leakage-control techniques here, the simpler temperature-dependent function given in Equation 7 seems adequate for our current work.

5.4. Benchmarks

We evaluate our results using benchmarks from the SPEC CPU2000 suite. The benchmarks are compiled and statically linked for the Alpha instruction set using the Compaq Alpha compiler with SPEC *peak* settings and include all linked libraries but no operating-system or multiprogrammed behavior. For each program, we fast-forward to a single representative sample of 500 million instructions. The location of this sample is chosen using the data provided by Sherwood *et al.* [38]. Simulation is conducted using SimpleScalar's EIO traces to ensure reproducible results for each benchmark across multiple simulations.

Due to the extensive number of simulations required for this study and the fact that many did not run hot enough to be interesting thermally, we used only 11 of the total 26 SPEC2k benchmarks. A mixture of integer and floating-point programs with low, intermediate, and extreme thermal demands were chosen; all those we omitted operate well below the $81.8°$ trigger threshold. Table 6 provides a list of the benchmarks we study along with their basic performance, power, and thermal characteristics. It can be seen that IPC and peak operating temperature are only loosely correlated with average power dissipation. For most SPEC benchmarks, and all those in Table 6, the hottest unit is the integer register file—interestingly, this is even true for most floating-point and memory-bound benchmarks. It is not clear how true this will be for other benchmark sets.

For the benchmarks that have multiple reference inputs, we chose one. For *perlbmk*, we used splitmail.pl with arguments "957 12 23 26 1014"; *gzip* - graphic; *bzip2* - graphic; *eon* - rushmeier; *vortex* - lendian3; *gcc* - expr; and *art* - the first reference input with "-startx 110".

5.5. Package, Warmup, and Initial Temperatures

The correct choice of convection resistance and heat-sink starting temperature are two of the most important determinants of thermal behavior over the relatively short time scales than can be tractably simulated using SimpleScalar.

To obtain a useful range of benchmark behaviors for studying dynamic thermal management, we set the convection resistance manually. We empirically determined a value of 0.8 K/W that yields the most interesting mix of behaviors. This

	IPC	Average Power (W)	FF (bil.)	% Cycles in Thermal Viol.	Dynamic Max Temp. (°C)	Steady-State Temp. (°C)	Sink Temp. (no DTM) (°C)	Sink Temp. (w/ DTM) (°C)
Low Thermal Stress (cold)								
parser (I)	1.8	27.2	183.8	0.0	79.0	77.8	66.8	66.8
facerec (F)	2.5	29.0	189.3	0.0	80.6	79.0	68.3	68.3
Severe Thermal Stress (medium)								
mesa (F)	2.7	31.5	208.8	40.6	83.4	82.6	70.3	70.3
perlbmk (I)	2.3	30.4	62.8	31.1	83.5	81.6	69.4	69.4
gzip (I)	2.3	31.0	77.3	66.9	84.0	83.1	69.8	69.6
bzip2 (I)	2.3	31.7	49.8	67.1	86.3	83.3	70.4	69.8
Extreme Thermal Stress (hot)								
eon (I)	2.3	33.2	36.3	100.0	84.1	84.0	71.6	69.8
crafty (I)	2.5	31.8	72.8	100.0	84.1	84.1	70.5	68.5
vortex (I)	2.6	32.1	28.3	100.0	84.5	84.4	70.8	68.3
gcc (I)	2.2	32.2	1.3	100.0	85.5	84.5	70.8	68.1
art (F)	2.4	38.1	6.3	100.0	87.3	87.1	75.5	68.1

Table 6. Benchmark summary. "I" = integer, "F" = floating point. Fast-forward distance (FF) represents the point, in billions of instructions, at which warmup starts (see Sec. 5.5).

represents a medium-cost heat sink, with a modest savings of probably less than \$10 [48] compared to the 0.7 K/W convection resistance that would be needed without DTM. Larger resistances, *e.g.* 0.85 K/W, save more money but give hotter maximum temperatures and less variety of thermal behavior, with all benchmarks either hot or cold. Smaller resistances save less money and bring the maximum temperature too close to 85° to be of interest for this study.

The initial temperatures that are set at the beginning of simulation also play a large role in thermal behavior. The most important temperature is that of the heat sink. Its time constant is on the order of several minutes, so its temperature barely changes and certainly does not reach steady-state in our simulations. This means simulations must begin with the correct heat-sink temperature, otherwise dramatic errors occur. For experiments with DTM (except TT-DFS), the heat-sink temperature should be set to a value commensurate with the maximum tolerated die temperature (81.8° with our sensor architecture): the DTM response ensures that chip temperatures never exceed this threshold, and heat sink temperatures are correspondingly lower than with no DTM. If the much hotter no-DTM heat-sink temperatures are used by mistake, we have observed dramatic slowdowns as high as 4.5X for simulations of up to one billion cycles, compared to maximum slowdowns of about 1.5X with the correct DTM heat-sink temperatures. The difference between the two heat-sink temperatures can be seen in Table 6. All our simulations use the appropriate values from this table.

Another factor that we have not accounted for is multi-programmed behavior. A "hot" application that begins executing when the heat sink is cool may not generate thermal stress before its time slice expires. Rohou and Smith [34] used this to guide processor scheduling and reduce maximum operating temperature.

Other structures will reach correct operating temperatures in simulations of reasonable length, but correct starting temperatures for all structures ensure that simulations are not influenced by such transient artifacts. This means that after loading the SimpleScalar EIO checkpoint at the start of our desired sample, it is necessary to warm up the state of large structures like caches and branch predictors, and then to literally warm up HotSpot. When we start simulations, we first run the simulations in full-detail cycle-accurate mode (but without statistics-gathering) for 100 million cycles to train the caches—including the L2 cache—and the branch predictor. This interval was found to be sufficient using the MRRL technique proposed by Haskins and Skadron [19], although a more precise use of this technique would have yielded specific warmup intervals for each benchmark. With the microarchitecture in a representative state, we deal with temperatures. These two issues must be treated sequentially, because otherwise cold-start cache effects would idle the processor and affect temperatures. To warm up the temperatures, we first set the blocks' initial temperatures to the steady-state temperatures calculated using the per-block average power dissipation for each benchmark. This accelerates thermal warmup, but a dynamic warmup phase is still needed because the sample we are at probably does not exhibit average behavior in all the units, and because this is the easiest way to incorporate the role of the temperature dependence of leakage on warmup. We therefore allow the simulation to continue in full-detail cycle-accurate mode for another 200 million cycles to allow temperatures to reach truly representative values. Only after these two warmup phases have completed do we begin to track any experimental statistics.

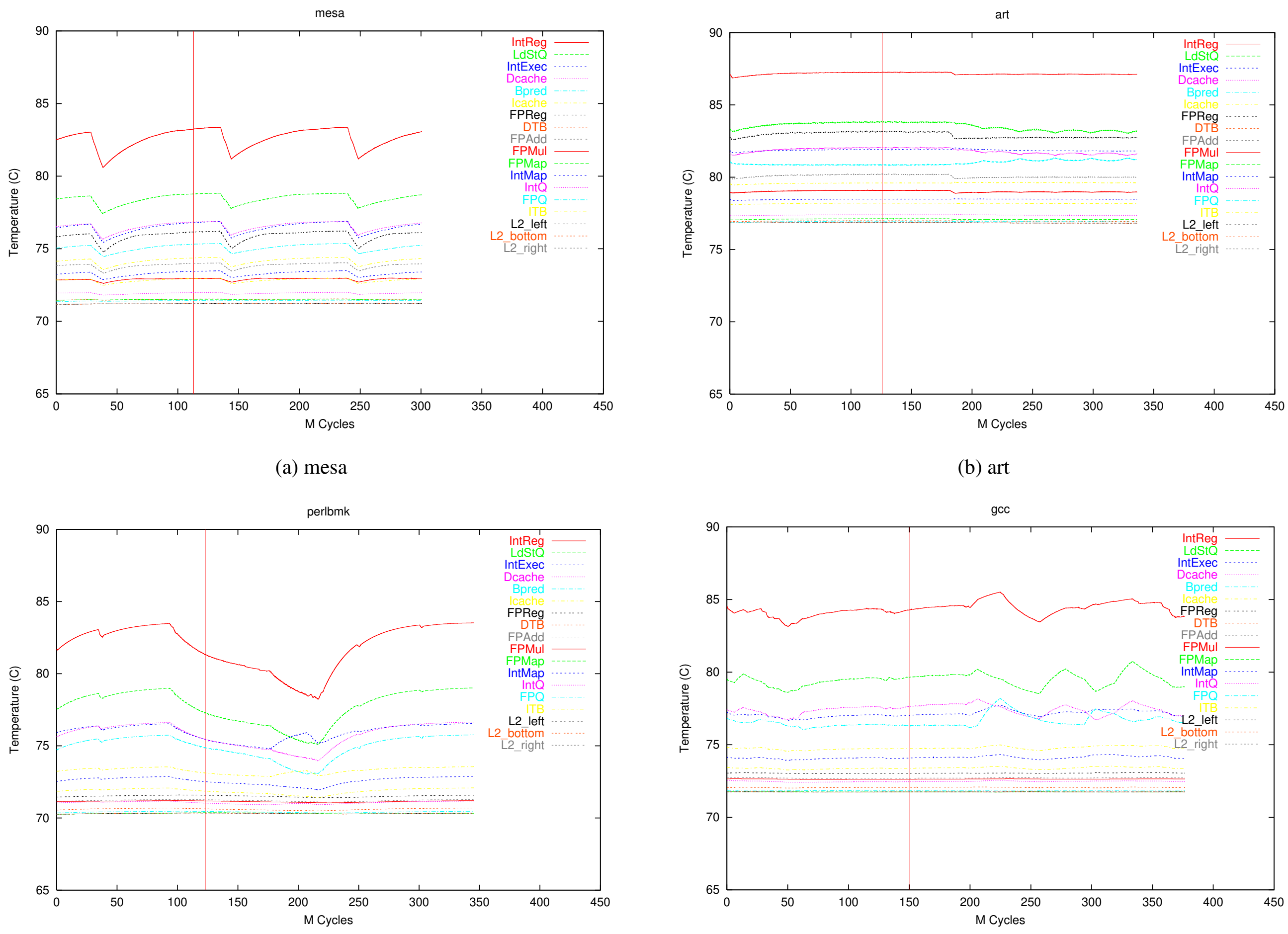
Note that, in order to have the statistics come from the program region that matches the Sherwood simulation points, the checkpoints must actually correspond to a point 300 million instructions prior to the desired simulation point. The “FF” column in Table 6 therefore shows where are checkpoints are captured, namely the fast-forward distance to reach the point where are warmup process begins.

5.6. Time Plots

To more clearly illustrate the time-varying nature of programs’ thermal behavior, in Figure 14 we present a few plots of programs’ operating temperature (with no DTM) in each unit as a function of time. In each plot, the vertical line toward the left side of the plot indicates when the warmup period ends.

Mesa (Figure 14a) deserves special comment because it shows clear program phases. At each drop in its sawtooth curve, we found (not shown) a matching sharp rise in L1 and L2 data misses and a sharp drop in branch mispredictions. The rate of rise and fall exactly matches what we calculate by hand from the RC time constants. The temperatures are only varying by a small amount near the top of their range. So the increase in temperature occurs slowly, like a capacitor that is already close to fully charged, and the decrease in temperature is quite sharp, like a full capacitor being discharged.

At the other end of the spectrum is *art*, which has steady behavior and therefore a flat temperature profile.



(a) mesa

(b) art

(c) perlbmk

(d) gcc

Figure 14. Operating temperature as a function of time (in terms of number of clock cycles) for various warm and hot benchmarks.

6. Results for DTM

In this section, we use the HotSpot thermal model to evaluate the performance of the various techniques described in Section 4. First we assume realistic, noisy sensors, and then consider how much the noise degrades DTM performance, presenting additional data that did not appear in the conference paper. The remainder of the section introduces new discussion and further results that expand beyond what the conference paper presented, exploring the MC technique, lateral thermal diffusion, and the role of initial heat-sink temperatures in further detail.

Note that since completing the conference version of this paper, we discovered a bug in the controller for DVS which caused it to engage overly aggressive voltage reductions. This paper presents the updated results. In contrast to the previous results, idealized DVS is now competitive with local toggling, and non-idealized DVS is now competitive with PI-global-toggling.

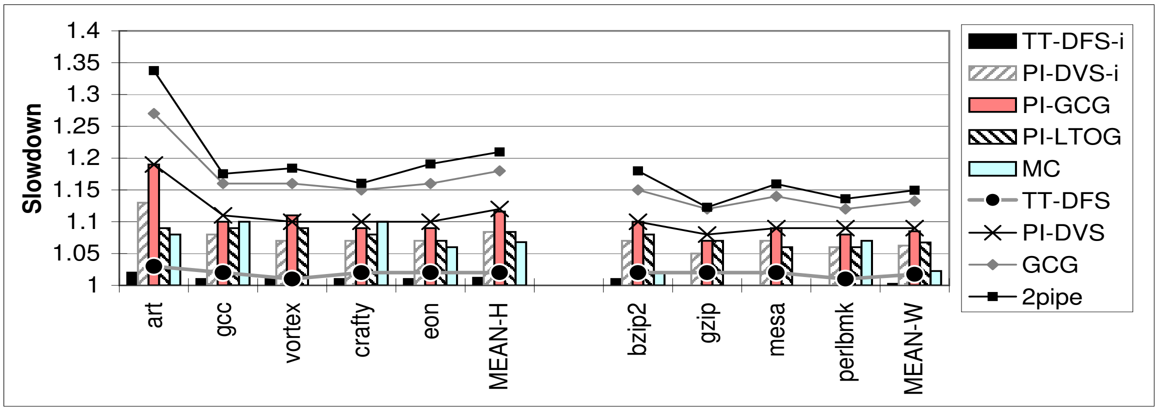


Figure 15. Slowdown for DTM. Bars: better techniques. Lines: weaker techniques.

6.1. Results with Sensor Noise Present

Figure 15 presents the slowdown (execution time with thermal management divided by original execution time) for the "hot" and "warm" benchmarks for each of the thermal management techniques. The bars are the main focus: they give results for the better techniques: "ideal" for TT-DFS and PI-DVS, the PI-controller version of GCG, PI local toggling, and MC. The lines give results for non-ideal TT-DFS and the weaker techniques: non-ideal PI-DVS, GCG with no controller (*i.e.*, all-or-nothing), and 2pipe. None of the techniques incur thermal violations. Only the hot and warm benchmarks are shown; the two cold benchmarks are unaffected by DTM, except for mild effects with TT-DFS (see below).

The best technique for thermal management by far is TT-DFS, with the TT-DFS-i version being slightly better. The performance penalty for even the hottest benchmarks is small; the worst is *art* with only a 2% slowdown for TT-DFS-i and a 3% slowdown for TT-DFS. The change in operating frequency also reduces power dissipation and hence slightly reduces the maximum temperature, bringing *art* down to 87.0°. If the maximum junction temperature of 85° is strictly based on timing concerns, and slightly higher temperatures can be tolerated without unduly reducing operating lifetime, then TT-DFS is vastly superior because its impact is so gentle.

It might seem there should be some benefit with TT-DFS from *increasing* frequency when below the trigger threshold, but we did not observe any noteworthy speedups—even for TT-DFS-i with the coldest benchmark, *mcf*, we observed only a 2% speedup, and the highest speedup we observed was 3%. With TT-DFS, a few benchmarks actually experienced a 1% slowdown, and the highest speedup we observed was 2%. The reason for the lack of speedup with DFS is partly that the slope is so small—this helps minimize the slowdown for TT-DFS with warm and hot benchmarks, but minimizes the benefit for cold ones. In addition, for higher frequency to provide significant speedup, the application must be CPU-bound, but then it will usually be hot and frequency cannot be increased.

If the junction temperature of 85° is dictated not only by timing but also physical reliability, then TT-DFS is not a viable approach. Of the remaining techniques, MC, idealized DVS, and PI-LTOG are the best. MC with a one-cycle penalty is best for all but three applications, *gcc, crafty*, and *perlbmk*, and the average slowdown for MC is 4.8% compared to 7.4% for DVS-i and 7.7% for PI-LTOG. Naturally, MC performs better if the extra communication latency to the spare register file is smaller: if that penalty is two cycles instead of one, MC's average slowdown is 7.5%. It is interesting to note that MC alone is not able to prevent all thermal violations; for two benchmarks, our MC technique engaged the fallback technique, PI-LTOG, and for those benchmarks spent 20-37% of the time using the fallback technique. This means that the choice of fallback technique can be important to performance. Results for these two benchmarks are much worse, for example, if we use DVS or GCG as the fallback.

Migrating computation and localized toggling outperform global toggling and non-idealized DVS, and provide similar performance as idealized DVS, even though DVS obtains a cubic reduction in power density relative to the reduction in frequency. The reason is primarily that GCG and DVS slow down the entire chip, and non-ideal DVS also suffers a great deal from the stalls associated with changing settings. In contrast, MC and PI-LTOG are able to exploit ILP.

A very interesting observation is that with MC, two benchmarks, *gzip* and *mesa*, never use the spare unit and suffer no slowdown, and *vortex* uses it only rarely and suffers almost no slowdown. The new floorplan by itself is sufficient to reduce thermal coupling among the various hot units in the integer engine and therefore prevents many thermal violations.

Although we were not able to explore a wider variety of floorplans, the success of these floorplan-based techniques suggests an appealing way to manage heat. And once alternate floorplans and extra computation units are contemplated, the interaction of performance and temperature for microarchitectural clusters [11] becomes an interesting area for further investigation. Our MC results also suggest the importance of modeling lateral thermal diffusion.

These results also suggest that a profitable direction for future work is to re-consider the tradeoff between latency and heat when designing floorplans, and that a hierarchy of techniques from gentle to strict—as suggested by Huang *et al.* [20]—is most likely to give the best results. A thermal management scheme might be based on TT-DFS until temperature reaches a dangerous threshold, then engage some form of migration, and finally fall back to DVS.

6.2. Role of Sensor Error

Sensor noise hurts in two ways; it generates spurious triggers when the temperature is actually not near violation, and it forces a lower trigger threshold. Both reduce performance. Figure 16 shows the impact of both these effects for our DTM techniques (for TT-DFS and DVS, we look at the non-ideal versions). The total height of each bar represents the slowdown with respect to DTM simulations with noise-free sensors and a trigger threshold of 82.8°. The bottom portion of each bar shows the slowdown from reducing the trigger by one degree while keeping the sensors noise-free, and the top portion shows the subsequent slowdown from introducing sensor noise of $\pm 1°$. (For the warm and hot benchmarks, the impact of both these sensor-related effects was fairly similar.)

For TT-DFS the role of the different threshold was negligible. That is because the TT-DFS change in frequency for one degree is negligible. MC also experiences less impact from the different threshold. We attribute this to the fact that the floorplan for MC itself has a cooling effect and reduces the need for DTM triggers. Otherwise, lowering the trigger threshold from 82.8° (which would be appropriate if noise were not present) reduces performance by 1–3% for the other major techniques. 2pipe experiences a larger impact—4%—because it is so inefficient at cooling that it must work harder to achieve each degree of cooling.

The spurious triggers further reduce performance by 0–2% for TT-DFS; 3–6% for PI-GCG; by 6–8% for PI-DVS, with *art* an exception for PI-DVS at 11%; by 2–5% for LTOG; 0–4% for MC; and 4–9% for 2pipe. The higher impact of noise for DVS is due to the high cost of stalling each time a spurious trigger is invoked, and similarly, the higher impact of noise for 2pipe is due to the cost of draining the pipeline each time a spurious trigger is invoked.

Sensor error clearly has a significant impact on the effectiveness of thermal management. With no sensor noise and a higher trigger, DTM overhead could be substantially reduced: TT-DFS's slowdown for the hot benchmarks moves from 1.8% to 0.8%, PI-DVS's slowdown from 10.7% to 2.5%, PI-GCG's slowdown from 11.6% to 3.6%, GCG's slowdown from 17.9% to 6.6%, PI-LTOG's slowdown from 8.4% to 5.2%, MC's slowdown from 7.0% to 4.2%, and 2pipe's slowdown from 21.0% to 9.5%. These results only considered the impact of sensor noise, the "S" factor. Reducing sensor offset—the "G" factor—due to manufacturing variations and sensor placement would provide substantial further improvements commensurate with the impact of the 1° threshold difference seen in the black portion of the bars.

Overall, our results also indicate not only the importance of modeling temperature in thermal studies, but also the importance of modeling realistic sensor behavior. And finding new ways to determine on-chip temperatures more precisely can yield substantial benefits.

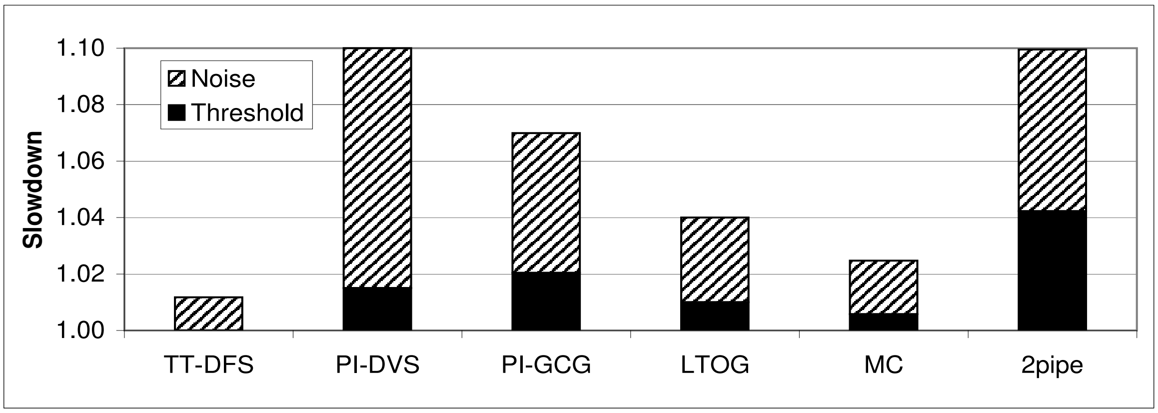


Figure 16. Slowdown for DTM from eliminating sensor noise, and from the consequent increase in trigger threshold to $82.8°$**.**

6.3. Further Analysis of MC

The MC technique, with local toggling as a fallback, merits further discussion in order to clarify the respective roles of floorplanning, migration, and the fallback technique.

If the MC floorplan is used without enabling the actual use of the migration, the spare register file is unused and has a mild cooling effect. The permutation of the floorplan also changes some of the thermal diffusion behavior. This has negligible effect for most benchmarks, and actually mildly exacerbates hotspots for *perlbmk*, but actually is enough to eliminate thermal violations for *gzip, mesa*, and *vortex*.

When the MC technique is enabled, it is able to eliminate thermal violations without falling back to local toggling in all but two benchmarks, *gcc* and *perlbmk*.

We erroneously stated in Section 4 of our conference paper that the "MC" technique incurs a two-cycle penalty on each access to the secondary register file. In fact, the results reported in that paper and in Figure 15 are for a one-cycle latency. As Figure 17 shows, a two-cycle latency significantly increases the cost of MC, from an average of 4.9% to 7.5%. On the other hand, we do not fully model the issue logic, which should factor in this latency and wake instructions up early enough to read the register file by the time other operands are available on the bypass network. This makes both sets of results (one and two cycle penalties) pessimistic.

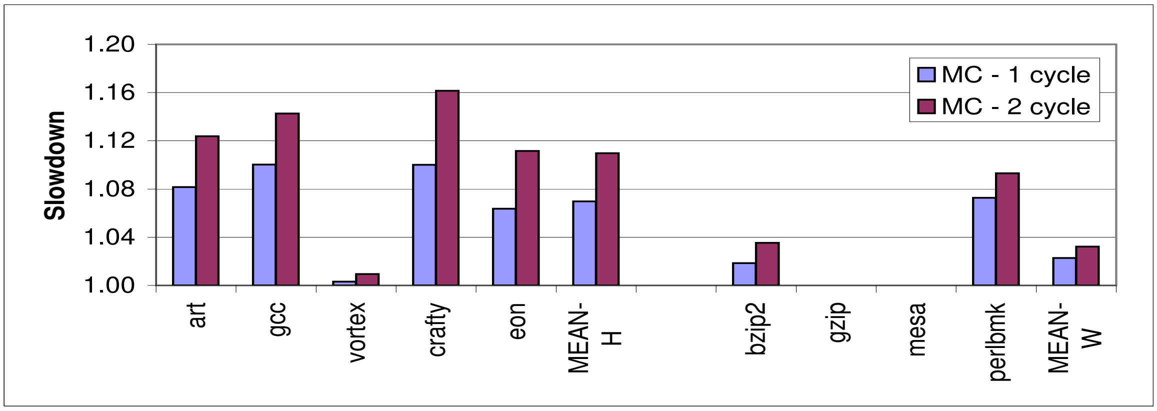


Figure 17. Slowdown for MC with 1- and 2-cycle penalties for accessing the spare register file.

Other floorplans that accommodate the spare register file may give different results, and spare copies of other units may be useful as well, especially for programs that cause other hotspots. We have not yet had a chance to explore these issues.

Another study we have not had a chance to perform is the cost-benefit analysis of whether the extra die area for the spare register file would be better used for some other structure, with purely local toggling as the DTM mechanism.

6.4. Importance of Modeling Lateral Thermal Diffusion

Our validation results in Section 3.4 (Figure 7) showed that modeling thermal diffusion with lateral thermal resistance yields significantly more accurate results than if the lateral heat flow is omitted. Here, we wish to clarify the importance of modeling lateral heat flow.

	loose-correct	tight-correct	loose-simple	tight-simple
art	1.00	1.00	1.00	1.00
gcc	1.00	1.00	1.00	1.00
vortex	1.00	1.00	1.00	1.00
crafty	1.00	1.00	1.00	1.00
eon	1.00	1.00	1.00	1.00
bzip2	0.68	0.76	0.90	0.90
gzip	0.67	0.72	0.91	0.91
mesa	0.42	0.58	0.71	0.71
perlbmk	0.31	0.36	0.39	0.39
facerec	0.00	0.00	0.00	0.00
parser	0.00	0.00	0.00	0.00

Table 7. Fraction of cycles in thermal violation (no DTM modeled) for the two different floorplans (loose and tight) with lateral thermal diffusion properly modeled (correct), and with lateral resistances omitted (simple).

	loose-correct	tight-correct	loose-simple	tight-simple
art	1.00	1.00	1.00	1.00
gcc	1.00	1.00	1.00	1.00
vortex	1.00	1.00	1.00	1.00
crafty	1.00	1.00	1.00	1.00
eon	1.00	1.00	1.00	1.00
bzip2	1.00	1.00	1.00	1.00
gzip	1.00	1.00	1.00	1.00
mesa	0.87	0.94	1.00	1.00
perlbmk	0.45	0.47	0.52	0.52
facerec	0.00	0.00	0.00	0.00
parser	0.00	0.00	0.00	0.00

Table 8. Fraction of cycles above the thermal trigger point (no DTM modeled) for the two different floorplans.

Lateral thermal diffusion is important for three reasons. First, it can influence the choice of a floorplan and the placement of spare units or clusters for techniques like MC or multi-clustered architectures. Second, it can have a substantial impact on the thermal behavior of individual units. When a consistently hot unit is adjacent to units that are consistently colder, the colder units help to draw heat away from the hot unit. Failing to model lateral heat flow in situations like these can make hot units look hotter than they really are, overestimating thermal triggers and emergencies and potentially distorting conclusions that might be drawn about temperature-aware design. Third, as shown in Section 3.4, failing to model lateral heat flow also produces artificially fast thermal rise and fall times, contributing to the overestimation of thermal triggers but also making DTM techniques seem to cool the hotspots faster than would really occur.

As a preliminary investigation of these issues, we compared the two floorplans shown in Figure 18. The "tight" floorplan in (a) places several hot units like the integer register file, integer functional units, and load-store queue near each other, while the "loose" one in (b) places the hottest units far from each other. In our experiments, we did not change any access latencies to account for distance between units in the two floorplans. This isolates thermal effects that are due to thermal diffusion rather than differences in access latency. We compared the thermal behavior of these floorplans using our full proposed model and also a modified version in which lateral thermal resistances have been removed.

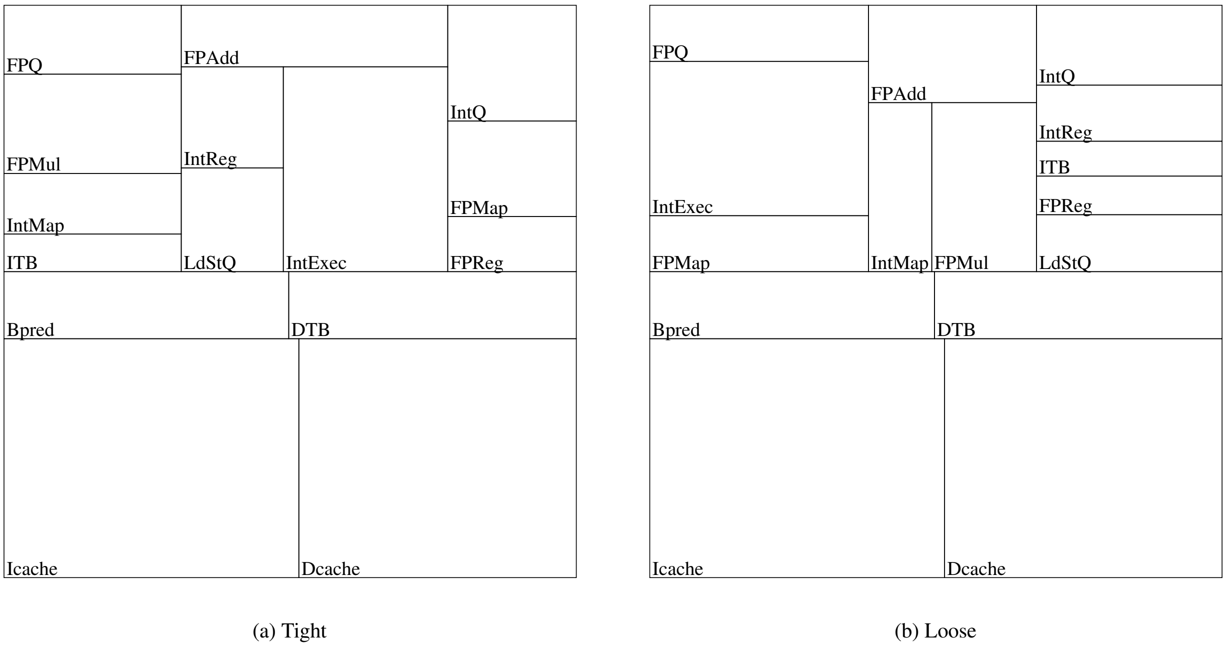


Figure 18. Two floorplans used to study effects of lateral thermal diffusion.

Table 7 presents, for each floorplan, the fraction of cycles spent in thermal violation (no DTM was used for these experiments). The left-hand pair of columns present data obtained with the full model ("correct"), and the right-hand pair of columns present data obtained with the lateral resistances omitted ("simple"). Table 8 presents the fraction of cycles spent above the thermal trigger temperature.

Looking at the correct data, the distinction between the two floorplans is clear, with the tight floorplan spending more time at higher temperatures due to the co-location of several hot blocks. The tight floorplan will engage DTM more. A time plot for the integer register file of *mesa* is given in Figure 19.

The simplified model, on the other hand, fails in two regards. First, it predicts higher temperatures and higher frequencies of thermal violation, higher even than what is observed with the tight floorplan. This happens because even the tight floorplan is able to diffuse away some of the heat in the hot blocks to neighboring blocks. This difference is largest for *gzip* and *mesa*. The artificially high temperatures mean that simulations of DTM will generate spurious thermal triggers and predict larger performance losses for DTM than would really be expected. Second, the failure to model lateral heat flow means that issues related to floorplan simply cannot be modeled, as seen by the fact that the two floorplans give identical results.

Without modeling lateral thermal diffusion, tradeoffs between thermal management and latency cannot be explored, and studies of dynamic thermal management may give incorrect results.

6.5. Role of Initial Heat-Sink Temperature

Finally, we wish to follow up on the point made in Section 5.5 that the choice of initial heat-sink temperature plays a major role, and the use of incorrect or unrealistic temperatures can yield dramatically different simulation results. Figure 20 plots the percentage error in execution time for various DTM techniques when the no-DTM heat-sink temperatures are used instead of the proper DTM heat-sink temperatures. The error only grows as the heat sink temperature increases. When we tried heat-sink temperatures in the 90°s, we observed slowdowns of as much as 4.5X.

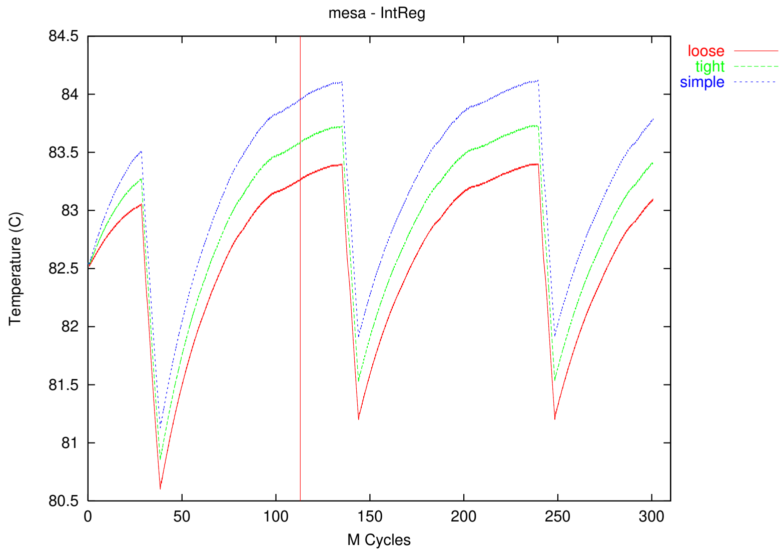


Figure 19. Temperature as a function of time for the integer register file with the mesa benchmark, for two different floorplans (tight and loose) and a simulation with lateral thermal resistance omitted (simple).

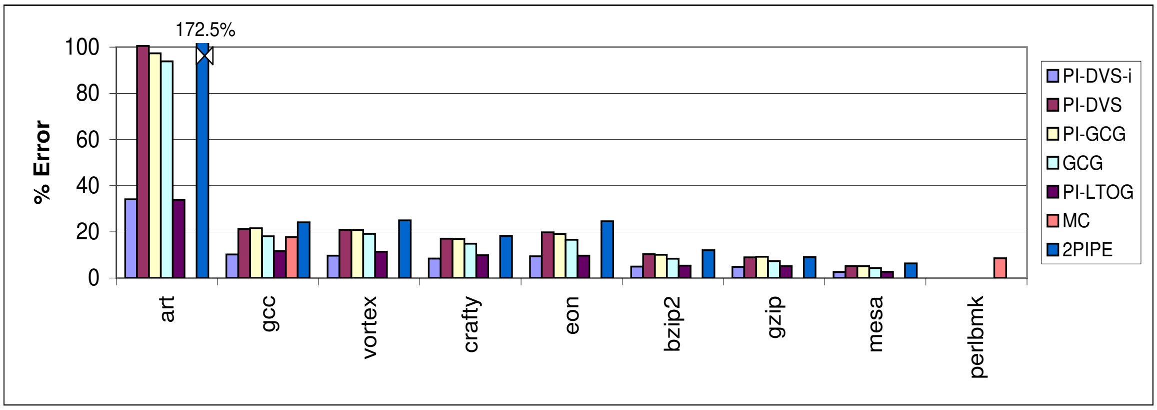


Figure 20. Percent error in execution time when DTM techniques are modeled using no-DTM heat sink temperatures.

The main reason this is an issue is that microarchitecture power/performance simulators have difficulty simulating a benchmark long enough to allow the heat sink to change temperature and settle at a proper steady-state temperature, because the time constant for the heat sink is so large. Over short time periods, changes in heat-sink temperature effectively act as an offset to the chip surface temperatures. That is why the correct steady-state temperature must be obtained before simulation begins. This means that for each DTM technique, floorplan, or trigger temperature, new initial temperatures must be determined. Developing simulation techniques or figures of merit to avoid this tedious task is an important area for future work. Fortunately, for all of our DTM techniques except MC, we found that the same initial "with-DTM" temperatures given in Table 6 were fine. MC's use of a different floorplan requires a separate set of initial temperatures.

Because time slices are much smaller than the time constant for the thermal package, a multiprogrammed workload will tend to operate with a heat-sink temperature that is some kind of average of the natural per-benchmark heat-sink temperatures, possibly reducing the operating temperature observed with the hottest benchmarks and conversely requiring DTM for cold benchmarks. Thermal behavior and the need for DTM will therefore depend on the CPU scheduling policy, which Rohou and Smith used to help regulate temperature in [34]. Combining architecture-level and system-level thermal management techniques in the presence of context switching is another interesting area for future work.

7. Conclusions and Future Work

This paper has presented HotSpot, a practical and computationally efficient approach to modeling thermal behavior in architecture-level power/performance simulators. Our technique is based on a simple network of thermal resistances and capacitances that have been combined to account for heating within a block due to power dissipation, heat flow among neighboring blocks, and heat flow into the thermal package. The model has been validated against finite-element simulations using Floworks, a commercial simulator for heat and fluid flow. HotSpot is publicly available at http://lava.cs.virginia.edu/hotspot.

Using HotSpot, we can determine which are the hottest microarchitectural units; understand the role of different thermal packages on architecture, performance, and temperature; understand programs' thermal behavior; and evaluate a number of techniques for regulating on-chip temperature. When the maximum operating temperature is dictated by timing and not physical reliability concerns, "temperature-tracking" frequency scaling lowers the frequency when the trigger temperature is exceeded, with average slowdown of only 2%, and only 1% if the processor need not stall during frequency changes. When physical reliability concerns require that the temperature never exceed the specification—85° in our studies—the best solutions we found were an idealized form of DVS that incurs no stalls when changing settings or a feedback-controlled localized toggling scheme (average slowdowns 7.4 and 7.7% respectively), and a computation-migration scheme that uses a spare integer register file (average slowdown 5–7.5% depending on access time to the spare register file). These schemes perform better than global clock gating, and as well as or better than the ideal feedback-controlled DVS, because the localized toggling exploits instruction-level parallelism while GCG and DVS slow down the entire processor.

A significant portion of the performance loss of all these schemes is due to sensor error, which invokes thermal management unnecessarily. Even with a mere $\pm 1°$ margin, sensor error introduced as much as 11% additional slowdowns, which accounted in some cases for as much as 80% of the total performance loss we observed.

We feel that these results make a strong case that runtime thermal management is an effective tool in managing the growing heat dissipation of processors, and that microarchitecture DTM techniques must be part of any temperature-aware system. But to obtain reliable results, architectural thermal studies must evaluate their techniques based on *temperature* and must include the effects of sensor noise as well as lateral thermal diffusion.

We hope that this paper conveys an overall understanding of thermal effects at the architecture level, and of the interactions of microarchitecture, power, sensor precision, temperature, and performance. This paper only touches the surface of what we believe is a rich area for future work. The RC model can be refined in many ways; it can also be extended to multiprocessor, chip-multiprocessor, and simultaneous multithreaded systems; many new workloads and DTM techniques remain to be explored; a better understanding is needed for how programs' execution characteristics and microarchitectural behavior determine their thermal behavior; and clever data-fusion techniques for sensor readings are needed to allow more precise temperature measurement and reduce sensor-induced performance loss. Another important problem is to understand the interactions among dynamic management techniques for active power, leakage power, current variability, and thermal effects, which together present a rich but poorly understood design space where the same technique may possibly be used for multiple purposes but at different settings. Finally, thermal adjacency was shown to be important, making temperature-aware floorplanning an important area of research.

Acknowledgments

This work is supported in part by the National Science Foundation under grant nos. CCR-0133634 and MIP-9703440, a grant from Intel MRL, and an Excellence Award from the Univ. of Virginia Fund for Excellence in Science and Technology. We would also like to thank Peter Bannon, Howard Davidson, Antonio González, Jose González González, Margaret Martonosi, and the anonymous reviewers for their helpful comments.

References

[1] K. Azar. Thermal design basics and calculation of air cooling limits, Oct. 2002. Tutorial at the 2002 International Workshop on THERMal Investigations of ICs and Systems (THERMINIC).
[2] A. Bakker and J. Huijsing. *High-Accuracy CMOS Smart Temperature Sensors*. Kluwer Academic, Boston, 2000.
[3] P. Bannon. Personal communication, Sep. 2002.

[4] W. Batty et al. Global coupled EM-electrical-thermal simulation and experimental validation for a spatial power combining MMIC array. *IEEE Transactions on Microwave Theory and Techniques*, pages 2820–33, Dec. 2002.

[5] Z. Benedek, B. Courtois, G. Farkas, E. Kollár, S. Mir, A. Poppe, M. Rencz, V. Székely, and K. Torki. A scalable multi-functional thermal test chip family: Design and evaluation. *Transactions of the ASME, Journal of Electronic Packaging*, 123(4):323–30, Dec. 2001.

[6] A. Bilotti. Static temperature distribution in IC chips with isothermal heat sources. *IEEE Transactions on Electron Devices*, ED-21:217–226, Mar. 1974.

[7] S. Borkar. Design challenges of technology scaling. *IEEE Micro*, pages 23–29, Jul.–Aug. 1999.

[8] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, pages 171–82, Jan. 2001.

[9] 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*, pages 83–94, June 2000.

[10] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.

[11] R. Canal, J.-M. Parcerisa, and A. González. A cost-effective clustered architecture. In *Proceedings of the 1999 International Conference on Parallel Architectures and Compilation Techniques*, pages 160–68, Oct. 1999.

[12] L. Cao, J. Krusius, M. Korhonen, and T. Fisher. Transient thermal management of portable electronics using heat storage and dynamic power disssipation control. *IEEE Transactions on Components, Packaging, and Manufacturing Technology—Part A*, 21(1):113–23, Mar. 1998.

[13] Y.-K. Cheng and S.-M. Kang. A temperature-aware simulation environment for reliable ULSI chip design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(10):1211–20, Oct. 2000.

[14] Compaq 21364 die photo. From website: CPU Info Center. http://bwrc.eecs.berkeley.edu/CIC/die_photos.

[15] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch. TEMPEST: A thermal enabled multi-model power/performance estimator. In *Proceedings of the Workshop on Power-Aware Computer Systems*, Nov. 2000.

[16] M. Fleischmann. Crusoe power management: Cutting x86 operating power through LongRun. In *Embedded Processor Forum*, June 2000.

[17] J. Garrett and M. R. Stan. Active threshold compensation circuit for improved performance in cooled CMOS systems. In *International Symposium on Circuits and Systems*, May 2001.

[18] S. Gunther, F. Binns, D. M. Carmean, and J. C. Hall. Managing the impact of increasing microprocessor power consumption. In *Intel Technology Journal*, Q1 2001.

[19] J. Haskins, Jr. and K. Skadron. Memory reference reuse latency: Accelerated sampled microarchitecture simulation. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 195–203, Mar. 2003.

[20] W. Huang, J. Renau, S.-M. Yoo, and J. Torellas. A framework for dynamic energy efficiency and temperature management. In *Proceedings of the 33rd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 202–13, Dec. 2000.

[21] Intel Corp. *Intel Pentium 4 Processor In the 423-pin Package: Thermal Design Guidelines*, Nov. 2000. Order no. 249203-001.

[22] A. Iyer and D. Marculescu. Power and performance evaluation of globally asynchronous locally synchronous processors. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 158–68, May 2002.

[23] V. Koval and I. W. Farmaga. MONSTR: A complete thermal simulator of electronic systems. In *Proceedings of the 31st Design Automation Conference*, June 1994.

[24] A. Krum. Thermal management. In F. Kreith, editor, *The CRC handbook of thermal engineering*, pages 2.1–2.92. CRC Press, Boca Raton, FL, 2000.

[25] S. Lee, S. Song, V. Au, and K. Moran. Constricting/spreading resistance model for electronics packaging. In *Proceedings of the ASME/JSME Thermal Engineering Conference*, pages 199–206, Mar. 1995.

[26] C.-H. Lim, W. Daasch, and G. Cai. A thermal-aware superscalar microprocessor. In *Proceedings of the International Symposium on Quality Electronic Design*, pages 517–22, Mar. 2002.

[27] R. Mahajan. Thermal management of CPUs: A perspective on trends, needs and opportunities, Oct. 2002. Keynote presentation at the 8th Int'l Workshop on THERMal INvestigations of ICs and Systems.

[28] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 132–41, June 1998.

[29] M. McManus and S. Kasapi. PICA watches chips work. *Optoelectronics World*, Jul. 2000.

[30] J. M. Rabaey. *Digital Integrated Circuits: A Design Perspective*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[31] M. Rencz and V. Székely. Studies on the error resulting from neglecting nonlinearity effects in dynamic compact model generation. In *Proceedings of the 8th Int'l Workshop on THERMal INvestigations of ICs and Systems*, pages 10–16, Oct. 2002.

[32] M. Rencz, V. Székely, A. Poppe, and B. Courtois. Friendly tools for the thermal simulation of power packages. In *Proceedings of the International Workshop On Integrated Power Packaging*, pages 51–54, July 2000.

[33] J. Robertson. Intel hints of next-generation security technology for mpus. *EE Times*, Sept. 10 2002.

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

[35] M.-N. Sabry. Dynamic compact thermal models: An overview of current and potential advances. In *Proceedings of the 8th Int'l Workshop on THERMal INvestigations of ICs and Systems*, Oct. 2002. Invited paper.

[36] H. Sanchez et al. Thermal management system for high-performance PowerPC microprocessors. In *COMPCON*, page 325, 1997.

[37] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 29–40, Feb. 2002.

[38] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.

[39] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical report, Compaq Western Research Laboratory, Feb. 2001.
[40] SIA. *International Technology Roadmap for Semiconductors*, 2001.
[41] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 17–28, Feb. 2002.
[42] K. Skadron, M. R. Stan, M. Barcella, A. Dwarka, W. Huang, Y. Li, Y. Ma, A. Naidu, D. Parikh, P. Re, G. Rose, K. Sankaranarayanan, R. Suryanarayan, S. Velusamy, H. Zhang, and Y. Zhang. Hotspot: Techniques for modeling thermal effects at the processor-architecture level. In *Proceedings of the 2002 International Workshop on THERMal Investigations of ICs and Systems (THERMINIC)*, pages 169–72, Oct. 2002.
[43] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, Apr. 2003.
[44] Standard Performance Evaluation Corporation. SPEC CPU2000 Benchmarks. http://www.specbench.org/osg/cpu2000.
[45] V. Székely, A. Poppe, A. Páhi, A. Csendes, and G. Hajas. Electro-thermal and logi-thermal simulation of VLSI designs. *IEEE Transactions on VLSI Systems*, 5(3):258–69, Sept. 1997.
[46] K. Torki and F. Ciontu. IC thermal map from digital and thermal simulations. In *Proceedings of the 2002 International Workshop on THERMal Investigations of ICs and Systems (THERMINIC)*, pages 303–08, Oct. 2002.
[47] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proceedings of the 2002 Workshop on Memory Performance Issues*, May 2002.
[48] R. Viswanath, W. Vijay, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, Q3 2000.
[49] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Technical Report CS-2003-05, University of Virginia Department of Computer Science, Mar. 2003.