Temperature-Aware Microarchitecture

[†]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 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, and that sensor imprecision has a substantial impact on the performance of DTM.*

1. Introduction

In recent years, power density in microprocessors has doubled every three years [3, 17], and this rate is expected to increase within one to two generations as feature sizes and frequencies scale faster than operating voltages [25]. 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 [3, 12], 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 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 [4, 12, 13] 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 [12], this can lead to dramatic savings. This is the philosophy behind the thermal design of the Intel Pentium 4 [12]. 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.

The Need for Architecture-Level Thermal Management. These *chip-level* hardware techniques illustrate both the benefits and challenges of runtime thermal management: while it can substantially reduce cooling costs and still allow typical applications

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

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 best 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 [20], 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 used to precisely regulate temperature while minimizing performance loss.

The Need for Architecture-Level Thermal Modeling. Some architectural techniques have already been proposed [4, 13, 16, 22, 26], 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.

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 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%. 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%. Local toggling performed almost as well, with a slowdown of 8%. All our experiments include the effects of sensor imprecision, which significantly handicaps runtime thermal management in current technology.

Because thermal constraints are becoming so severe, we expect that temperature-aware computing will be a rich area for research. 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.

The next section provides further background and related work. Then Section 3 describes our proposed model 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. Section 6 compares the various thermal-management techniques' ability to regulate temperature, and Section 7 concludes the paper. We have also published an extended version of this paper as a technical report [27], providing additional discussion and results.

2. Background and Related Work

Power densities are actually expected to rise faster in future technologies, because 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) [25] 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. 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. 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.

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 [21].

Despite the long-standing concern about thermal effects, only a few studies have been published in the architecture field. Gunther *et al.* [12] describe the thermal design approach for the Pentium 4, where thermal management is accomplished via global clock gating. Lim *et al.* [16] 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.

Huang *et al.* [13] 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 [4] 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 [22]), and speculation control [18]). 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. 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 3.6, this value does not track temperature reliably. A further problem is that, because no model of localized heating was available at the time, some of these techniques do not reduce power density in the actual hot spots.

Our prior work [26] 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.* [9]. 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. 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

There exists a well-known duality [14] between heat transfer and electrical phenomena. Heat flow can be described as a "current" passing through a thermal resistance and 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. The thermal Rs and Cs together lead to exponential rise and fall times characterized by thermal RC time constants analogous to the electrical RC 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*—*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.

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. 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 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.

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.

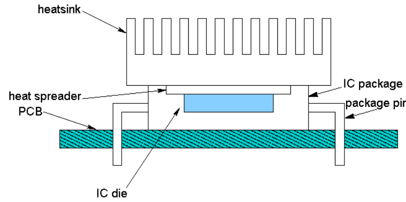


Figure 1. Side view of a typical package.

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

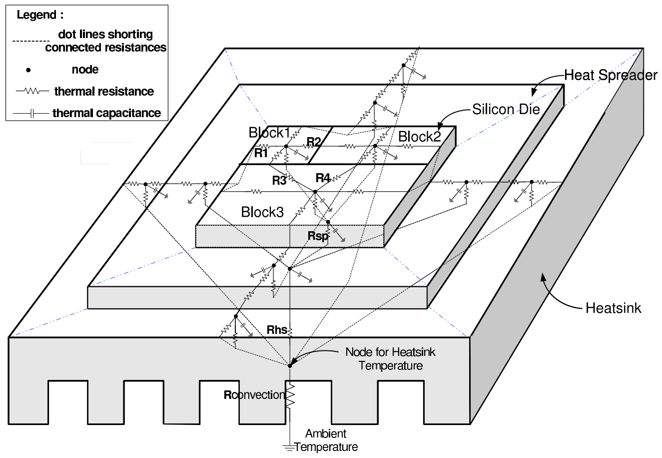


Figure 2. Example HotSpot RC model for a floorplan with three architectural units, a heat spreader, and a heat sink.

single thermal resistance ($R_{convection}$). Air is assumed to be at a fixed ambient temperature, which is generally assumed in thermal design to be 45°C [17] (this is not the room ambient, but the temperature inside the computer "box"). 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. 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.

3.2. Deriving the Model

In this section we sketch how the values of R and C are computed. A more detailed discussion can be found in the extended version of this paper [27].

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:

$$R = \frac{t}{k \cdot A} \tag{1}$$

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:

$$C = c \cdot t \cdot A \tag{2}$$

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. Typical chip thicknesses are in the range of 0.3–0.9mm; this paper studies a chip of 0.5mm thickness. 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.

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 [15]. 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. In the interests of space, details can be found in [27].

Normally, the package-to-air resistance ($R_{convection}$) would be calculated from the specific heat-sink configuration. In this paper, we instead manually choose a value resistance that gives us a good distribution of benchmark behaviors; see Section 5.4.

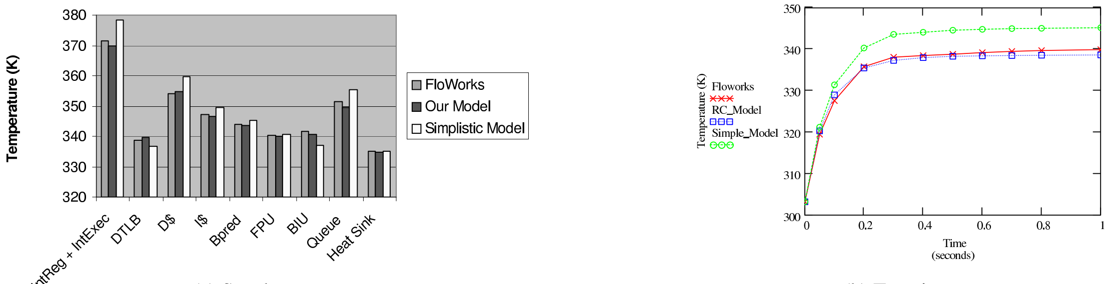
Capacitors in silicon must be multiplied by an empirical fitting constant of approximately 2. The reason for this is that the bottom surface of the chip is not actually isothermal, whereas HotSpot treats the bottom surface as such by feeding all vertical Rs for the die into a single node. The true isothermal surface lies somewhere in the heatsink, which means that the "equivalent thermal mass" that determines the rate of on-die heating is larger than the die thickness and makes the effective thermal capacitance larger than what Equation 2 yields. For a 0.5mm-thick die and reasonable distributions of power dissipation on the chip, we empirically determined the scaling factor remains close to 2 for reasonable floorplans and power dissipations by comparing to a reference finite-element heat-transport model (see Section 3.3). We chose this simplification because it simplifies the solution of the RC network without sacrificing accuracy of the temperatures on the die surface where the hottest temperatures occur. Nevertheless, because this factor is empirical and dependent on die thickness, it violates our goal of a fully parameterized model. Improving the model or developing an analytical expression is another area for future work.

Capacitors in the spreader and sink must be multiplied by about 0.4. Unlike the scaling for silicon, this is not an ad-hoc factor, but rather a consequence of using a simple single-lump C for each block, rather than a distributed model. The scaling factor we determine empirically does match the value predicted by [19].

HotSpot dynamically generates the RC circuit when initialized with a configuration that consists of the blocks' layout and their areas (see Section 3.4). 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 four iterations, 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.

3.3. Calibrating the Model

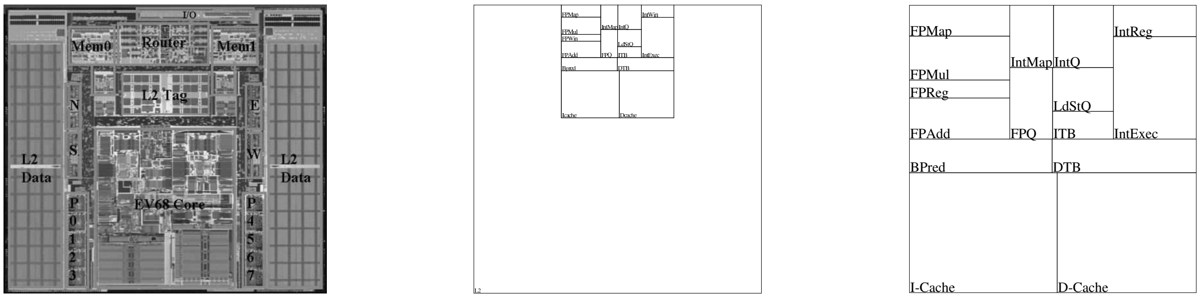
Dynamic compact models are well established in the thermal-engineering community, although we are unaware of any that have been developed to describe transient heat flow at the granularity of microarchitectural units. Of course, the exact compact model



(a) Steady state

(b) Transient

Figure 3. 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 4. (a): Die photo of the Compaq Alpha 21364 [8]. (b): Floorplan corresponding to the 21364 that is used in our experiments. (c): Closeup of 21364 core.

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.

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 silicon die with various floorplans and various power densities in each block; this die is attached to a copper spreader and a copper heat sink on one side, and covered by an insulating cap on the other side. Air is assumed to stay fixed at 45°C; airflow is laminar at 10 m/s. More detail on our Floworks modeling can be found in [27].

Floworks and HotSpot are only semi-independent, because the HotSpot fitting factor for the thermal capacitances in silicon was determined empirically using Floworks. Nevertheless, we can verify that the two obtain similar steady-state operating temperatures and transient response. Figure 3a 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 3b 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.

3.4. 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 4. 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 4a, 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.

3.5. 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.

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.

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.

Units	Temp. (°C)	Power (W)	Corr. coeff.
IntExec	77.3	4.45	0.09
IntReg	84.5	3.34	0.11
IntQ	72.5	0.23	0.05
IntMap	74.2	0.67	0.09
LdStQ	79.6	2.04	0.09
D-Cache	77.5	9.04	0.09
D-TLB	72.1	0.11	0.03
FPMap	71.7	0.02	0.14
FPAdd	72.7	0.38	0.00
FPQ	71.9	0.01	0.00
FPReg	73.1	0.24	0.01
L2	71.8	3.41	0.04

Table 1. Correlation of power and temperature

3.6. 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, 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.

To show the importance of using a thermal model instead of a power metric, we have computed the correlation coefficient between temperature and the moving average of power dissipation. The data in Table 1 come from *gcc*, a representative benchmark. Temperatures were collected using HotSpot, and power measurements were collected using various averaging periods for a simulation of 500 million cycles. We consistently found very poor correlations, especially in the hottest blocks. Some correlation coefficients for temperature and average power are reported in Table 1 for a power-averaging interval of one million cycles.

4. Techniques for Architectural DTM

This section describes the various architectural mechanisms for dynamic thermal management that are evaluated in this paper, 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 two techniques that have previously been proposed, namely DVS (but unlike prior work, we add feedback control) and global clock gating (where we also add feedback control).

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 [28], 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.*:

$$u[k] = u[k-1] + K_c \cdot e[k-1] \tag{3}$$

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 [4] 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 [11] 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 [23].

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. Note 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 [26] 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 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.

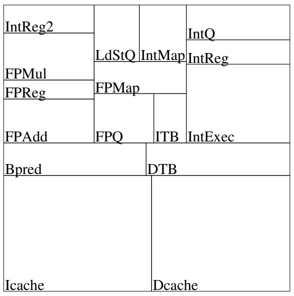


Figure 5. Floorplan with spare integer register file for migrating computation.

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 5. 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 two extra cycle for every register-file access. (For simplicity in our simulator, we approximate this by simply increasing the latency of every functional unit by two cycles, 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.

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.* [16] 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; results appear in [27]. MC could also be considered a limited form of multi-clustered architecture [7].

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 [4, 13], and is used for this purpose in Transmeta's Crusoe processors [10]. 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.

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. To prevent this, we apply a low-pass filter to the controller output.

Global Clock Gating. Finally, as a baseline, we consider global clock gating ("GCG") similar to what the Pentium 4 employs [12], 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 [4], 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. We only report results for global clock gating because GCG slightly outperforms plain fetch toggling, and PI-GCG slightly outperforms a PI version of fetch toggling. See [27].

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 [1]. The output current is digitized using a ring oscillator or some other type of delay element to produce a square wave that can 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 $\pm 2^{\circ}$C and sampling rate of 10 microseconds. This matches the results in [1]. We place one sensor per architectural block.

We model the imprecision by randomizing the true temperature reading over the specified range $\pm 2^{\circ}$. 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 this change, which is potentially as much as 0.4° if temperatures can rise 0.1° per $30\mu s$. For $\pm 2^{\circ}$, we are therefore able to reduce the uncertainty to $S = \frac{2}{\sqrt{10}} + 0.4 = \pm 1^{\circ}$. 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 $G = 2^{\circ}$.

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 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. 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. Further details can be found in the extended version of this paper [27].

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 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.

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.

5.2. Power-Performance Simulator

We use a power model based on power data for the Alpha 21364 [2]. 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 4, 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 [5] to provide a framework for integrating our power data with the underlying SimpleScalar [6] 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. 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.

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. 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). 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.

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 [30] 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 [25].

5.3. 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. 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.* [24]. 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 2 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 2, the hottest unit is the integer register file—interestingly, this is even true for most benchmarks, including floating-point and memory-bound benchmarks. It is not clear how true this will be for other benchmark sets.

5.4. 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 represents a medium-cost heat sink, with a modest savings of probably less than $10 [29] 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, dramatic slowdowns as high as 4.5X are observed 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 2. 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 [20] 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

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

Table 2. Benchmark summary. "I" = integer, "F" = floating point.

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. 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 we calculate using the per-block average power dissipation for each benchmark. This accelerates thermal warm up, but a dynamic warmup phase is still needed because the sample we are at probably does not exhibit average behavior in all the units 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.

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.

6.1. Results with Sensor Noise Present

Figure 6 presents the slowdown (execution time with thermal management) / (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 version of each technique: "ideal" for TT-DFS and PI-DVS, the PI-controller version of GCG, PI local toggling, and MC. The lines give results for the weaker version of some techniques: TT-DFS and PI-DVS with stalls for changing settings, and GCG with no controller (*i.e.*, all-or-nothing). None of the techniques incur thermal violations. Only the hot and warm benchmarks are shown; the two cold benchmarks are unaffected by DTM, except they obtain a 1% speedup with TT-DFS-i and a 1% slowdown with TT-DFS. We also tried some other cold benchmarks but observed minimal speedup with TT-DFS-i, and often slowdown with TT-DFS.

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. 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. For higher frequency to provide significant speedup, the application must be CPU-bound, but then it will 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 and PI-LTOG are the best. MC is better than PI-LTOG for all but three applications, *gcc, crafty*, and *perlbmk*, and the average slowdown for MC is 7% compared to 8% for PI-LTOG. Naturally, MC performs better if the extra communication latency to the spare register file is smaller: if that penalty is one cycle instead of two, MC's average slowdown is 5%. It is important to note that MC alone is not able to prevent all thermal violations; our MC technique spent 20-37% of its time using the fallback technique, PI-LTOG. This means that the choice of fallback technique is important to performance. Results are much worse, for example, if we use DVS or GCG as the fallback.

Migrating computation and localized toggling outperform global toggling and 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, although 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.

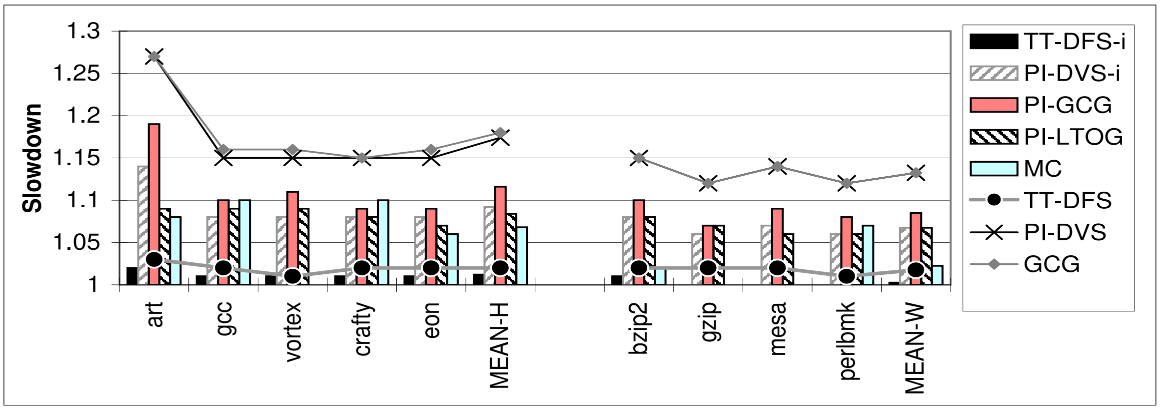


Figure 6. Slowdown for DTM. Bars: better version of technique. Lines: weaker version.

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 [7] becomes an interesting area for further investigation. 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.* [13]—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.

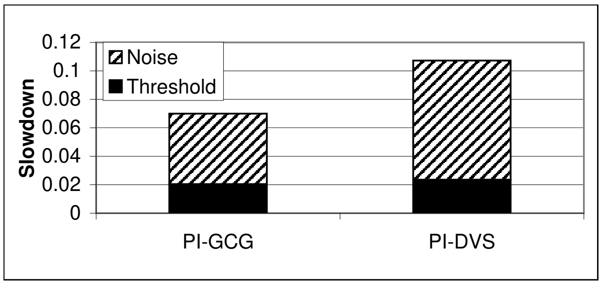


Figure 7. Slowdown for DTM from eliminating sensor noise, and from the consequent increase in trigger threshold to $82.8°$.

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 7 shows the impact of both these effects for the PI-GCG and PI-DVS techniques. The bottom portion of each bar shows the slowdown from only reducing the trigger by one degree, 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 effects was fairly similar. Lowering the trigger threshold from $82.8°$ (which would be appropriate if noise were not present) reduces performance by 1–4% for both PI-GCG and PI-DVS. The spurious triggers further reduce performance by 3–6% for PI-GCG, and by 6–9% for PI-DVS, with *art* an exception for PI-DVS at 13%. The higher impact of noise for DVS compared to GCG is due to the high cost of stalling each time a spurious trigger is invoked.

Sensor error clearly has a significant impact on the effectiveness of thermal management. These results only considered the impact of sensor noise, the "S" factor discussed in Section 4.2; with no sensor noise and a higher trigger, PI-GCG's slowdown for the hot benchmarks moves from 11.6% to 3.6%, and PI-DVS's slowdown from 17.4% to 5.2%. Reducing sensor offset—the "G" factor—due to manufacturing variations and sensor placement would provide substantial further improvements.

Overall, our results also indicate not only the importance of modeling temperature in thermal studies, but also the importance of modeling realistic sensor behavior.

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 with 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 solution we found was either a feedback-controlled localized toggling scheme (slowdown 8%), or a computation-migration scheme that uses a spare integer register file (slowdown 5–7% depending on access time to the spare register file). These schemes perform even better than global clock gating or an ideal feedback-controlled version of DVS that incurs no stalls when changing settings, 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 $\pm 1^\circ$ margin, sensor error introduced 6–13% slowdowns, nearly 75% 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.

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] A. Bakker and J. Huijsing. *High-Accuracy CMOS Smart Temperature Sensors*. Kluwer Academic, Boston, 2000.

[2] P. Bannon. Personal communication, Sep. 2002.

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

[4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proc. HPCA-7*, pp. 171–82, Jan. 2001.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. ISCA-27*, pp. 83–94, June 2000.

[6] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *ACM SIGARCH CAN*, 25(3):13–25, June 1997.

[7] R. Canal, J.-M. Parcerisa, and A. González. A cost-effective clustered architecture. In *Proc. PACT*, pp. 160–68, Oct. 1999.

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

[9] A. Dhodapkar, C. H. Lim, G. Cai, and W. R. Daasch. TEMPEST: A thermal enabled multi-model power/performance estimator. In *Proc. PACS*, Nov. 2000.

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

[11] J. Garrett and M. R. Stan. Active threshold compensation circuit for improved performance in cooled CMOS systems. In *Proc. ISCAS*, May 2001.

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

[13] W. Huang, J. Renau, S.-M. Yoo, and J. Torellas. A framework for dynamic energy efficiency and temperature management. In *Proc. Micro-33*, pp. 202–13, Dec. 2000.

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

[15] S. Lee, S. Song, V. Au, and K. Moran. Constricting/spreading resistance model for electronics packaging. In *Proc. AJTEC*, pp. 199–206, Mar. 1995.

[16] C.-H. Lim, W. Daasch, and G. Cai. A thermal-aware superscalar microprocessor. In *Proc. ISQED*, pp. 517–22, Mar. 2002.

[17] R. Mahajan. Thermal management of CPUs: A perspective on trends, needs and opportunities, Oct. 2002. Keynote presentation, THERMINIC-8.

[18] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. In *Proc. ISCA-25*, pp. 132–41, June 1998.

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

[20] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *Proc. FDDO-2*, Nov. 1999.

[21] M.-N. Sabry. Dynamic compact thermal models: An overview of current and potential advances. In *Proc. THERMINIC-8*, Oct. 2002.

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

[23] G. Semeraro et al. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proc. HPCA-8*, pp. 29–40, Feb. 2002.

[24] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proc. PACT*, Sept. 2001.

[25] SIA. *International Technology Roadmap for Semiconductors*, 2001.

[26] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proc. HPCA-8*, pp. 17–28, Feb. 2002.

[27] K. Skadron et al. Temperature-aware microarchitecture: Extended discussion and results. Tech. Report CS-2003-08, U.Va. Dept. of Computer Science, Apr. 2003.

[28] S. Velusamy, K. Sankaranarayanan, D. Parikh, T. Abdelzaher, and K. Skadron. Adaptive cache decay using formal feedback control. In *Proc. WMPI-2*, May 2002.

[29] R. Viswanath, W. Vijay, A. Watwe, and V. Lebonheur. Thermal performance challenges from silicon to systems. *Intel Tech. J.*, Q3 2000.

[30] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Tech. Report CS-2003-05, U.Va. Dept. of Computer Science, Mar. 2003.