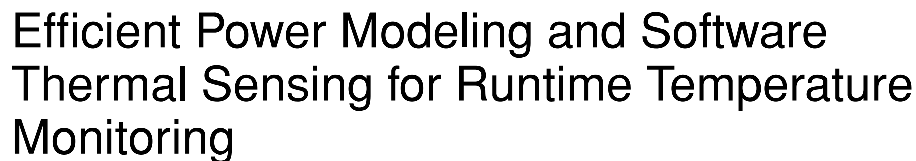

Efficient Power Modeling and Software Thermal Sensing for Runtime Temperature Monitoring

WEI WU
University of California at Riverside
LINGLING JIN
Nvidia Corporation
JUN YANG
University of Pittsburgh
and
PU LIU and SHELDON X.-D. TAN
University of California at Riverside

The evolution of microprocessors has been hindered by increasing power consumption and heat dissipation on die. An excessive amount of heat creates reliability problems, reduces the lifetime of a processor, and elevates the cost of cooling and packaging considerably. It is therefore imperative to be able to monitor the temperature variations across the die in a timely and accurate manner.

Most current techniques rely on on-chip thermal sensors to report the temperature of the processor. Unfortunately, significant variation in chip temperature both spatially and temporally exposes the limitation of the sensors. We present a compensating approach to tracking chip temperature through an OS resident software module that generates live power and thermal profiles of the processor. We developed such a software thermal sensor (STS) in a Linux system with a Pentium 4 Northwood core. We employed highly efficient numerical methods in our model to minimize the overhead of temperature calculation. We also developed an efficient algorithm for functional unit power modeling. Our power and thermal models are calibrated and validated against on-chip sensor readings, thermal images of the Northwood heat spreader, and the thermometer measurements on the package. The resulting STS offers detailed power and temperature breakdowns of

This work was supported in part by NSF Grant CCF-0541456. The work of S. X.-D. Tan and P. Liu was also supported by NSF CAREER Award CCF-0448534.
Authors' addresses: W. Wu, Department of Computer Science and Engineering, P. Liu, S. X.-D. Tan, Department of Electrical Engineering, University of California at Riverside, Riverside, CA 92521; L. Jin, Nvidia Corp., 2701 San Tomas Expy., Santa Clara, CA 95050; J. Yang (contact author), Electrical and Computer Engineering Department, University of Pittsburgh, Pittsburgh, PA 15261; email: junyang@ece.pitt.edu.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2007 ACM 1084-4309/2007/08-ART25 $5.00 DOI 10.1145/1255456.1255462 http://doi.acm.org/10.1145/1255456.1255462

each functional unit at runtime, enabling more efficient online power and thermal monitoring and management at a higher level, such as the operating system.

Categories and Subject Descriptors: C.1.1 [**Processor Architectures**]: Single Data Stream Architectures

General Terms: Design, Implementation

Additional Key Words and Phrases: Power, thermal

ACM Reference Format:

Wu, W., Jin, L., Yang, J., Liu, P., and Tan, S. X.-D. 2007. Efficient power modeling and software thermal sensing for runtime temperature monitoring. ACM Trans. Des. Autom. Electron. Syst. 12, 3, Article 25 (August 2007), 29 pages. DOI = 10.1145/1255456.1255462 http://doi.acm.org/10.1145/1255456.1255462

1. INTRODUCTION

As technology size enters the nanometer regime, power density has become one of the major constraints to attainable processor performance. High temperatures jeopardize chip reliability and significantly impact its performance. The immense spatial and temporal variation of chip temperature also creates great challenges to cooling and packaging, which are designed for typical, not worst-case, thermal conditions for cost effectiveness [Intel 2002]. This entails dynamic thermal management (DTM) to regulate chip temperature at runtime.

There has been plenty of research on DTM at the microarchitecture level [Brooks and Martonosi 2001; Gunther et al. 2001; Heo et al. 2003; Li et al. 2005b; Monferrer et al. 2005; Skadron et al. 2002, 2003; Srinivasan and Adve 2003]. Recently, a number of works have also shown the great potential of DTM performed at a higher level such as the operating system [Donald and Martonosi 2006; Powell et al. 2004; Kursun et al. 2006]. Such approaches require reliable and timely information about the thermal status of the entire chip, and hence leverage on-chip thermal sensors for temperature readings. However, DTM at high levels seeks to know the slopes of temperature transitions [Donald and Martonosi 2006], that is, the future temperatures in a short time span, but these depend on factors like future power, lateral heat diffusion, and the duration of the time span. Sensors alone are unaware of these factors and hence limited to providing helpful temperature information for high-level DTM.

In this article, we present a software approach for runtime temperature tracking to compensate for sensor deficiency. The software resides in an operating system kernel and probes existing hardware resources such as performance counters to calculate power and then the temperature at runtime. The software provides a framework for projecting future temperatures while considering lateral heat diffusion and the physical parameters of the processor package. Though there exist thermal estimation software tools in architecture simulators [Skadron et al. 2003], developing a similar tool for online temperature monitoring is challenging, as the main constraint is its overhead in time and space. Moreover, chip temperature varies with different packaging technologies and the physical conditions around the chip. Therefore, it is imperative

to calibrate the physical parameters used in the software for higher precision in temperature calculation.

This article makes the following contributions:

— We present a software thermal sensor (STS) that has been developed in a Linux 2.6.9 system with a Pentium 4 Northwood core. The STS consists of two main components: power measurement and temperature estimation, the latter dependent on the former. The power measurement leverages the *performance counters* existing in many contemporary processors to extract live activities of different components [Isci and Martonosi 2003]. We introduce a systematic methodology [Wu et al. 2006] to derive component powers because their accuracy determines the thermal results. Our methodology can be automated to make the framework of power measurement portable to different processor platforms.
— We adopt a highly efficient numerical method to solve temperatures from our thermal model in order to make it suitable for runtime implementation [Li et al. 2005a; Liu et al. 2005]. Recent fast thermal analyses mainly focus at the circuit or gate level, such as thermal ADI multigrid methods. They are not suitable in our context, which takes power information and calculates temperatures at runtime. Our thermal model is adapted from the well studied HotSpot thermal model [Skadron et al. 2003; Huang et al. 2005, 2004] with the floorplan of the processor under testing. Our efficient numerical method achieves a speedup of four orders of magnitude compared with the HotSpot tool. The computed temperature results are within a 0.08°C difference from the HotSpot for 22 tested SPEC2K benchmarks. We also included leakage power modeling, and considered the impact of environmental temperature changes, namely, the ambient air temperature.
— The important thermal resistances and capacitances within our thermal model are calibrated and validated against on-chip thermal sensor (only one in Pentium 4) readings, thermal images of the processor's heat spreader, and temperature measurements on the package. The complete STS is tested continuously for a duration of 8 hours on 22 SPEC2K benchmarks executed repeatedly to show that the errors due to modeling and abstraction do not propagate. The runtime performance overhead of the STS is less than 3% for a sampling interval of 10 ms.
— We present extensive thermal profiles on 22 SPEC2K benchmarks. Our observations prove the previous results and also uncover interesting new thermal features that had been difficult to obtain using simulations or simplified thermal modeling.

The remainder of the article is organized as follows. Section 2 discusses closely related work on power and thermal modeling. Section 3 introduces our software thermal sensor, consisting of power and temperature measuring components. Section 4 describes our methodology in calibrating our thermal model. Sections 5 and 6 show the power results and thermal profiles collected by our STS and interesting observations we made. In Section 8, we discuss the sources

of error of our STS, its limitations, and how it can be improved. Finally, Section 10 concludes.

2. RELATED WORK

Performing runtime power monitoring using performance counters has been demonstrated to be very effective [Bellosa 2000; Isci and Martonosi 2003; Joseph and Martonosi 2001; Seng and Tullsen 2003]. Counters provided by high-performance processors such as Pentium and UltraSPARC can be queried at runtime to derive the activities of each functional unit (FU). When combined with FU per-access power, counters can generate the dynamic and total power of the chip. Previous work either did not consider the leakage power or used a constant as a proxy, since the temperature was difficult to obtain. When the processor is at a high temperature, its leakage can contribute significantly to the total power [Huang et al. 2005]. We consider this as an integral part of our power estimation. Also, early approaches used microbenchmarks—handwritten code that exercises a minimum subset of specific FUs—to derive each FU's per-access power. We found this can lead to considerable deviations when tested with complex programs. Therefore, we develop a more systematic method to solve for a more dependable set of FU power. This is important to ensure the accuracy of the temperatures calculated in our STS.

Thermal modeling for microprocessors evolved from a coarse-grained measure for the entire chip [Dhodapkar et al. 2000] to a per-unit measure at the functional-unit level [Skadron et al. 2002], and lastly (and most recently) a detailed, more accurate multiunit measure, namely, HotSpot, which includes localized heating, thermal diffusion, coupling with the thermal package, leakage, and the thermal interface material modeling [Huang et al. 2005, 2004; Skadron et al. 2003]. The model takes as input FUs' power consumption and generates their temperatures. The entire tool set has been used in *simulation* intensively to model and study the thermal behavior and dynamic thermal management techniques. We carry this well-tuned model into a runtime environment by overcoming its performance barrier (discussed next) and further calibrating model parameters to come close to a real processor package.

Runtime temperature tracking in software has also been explored before [Bellosa et al. 2003; Lee and Skadron 2005; Weissel and Bellosa 2004]. In Bellosa et al. [2003] and Weissel and Bellosa [2004], the total power of the processor is first obtained using performance counters. Then, an analytical model is used, taking into account the heat sink to compute the chip temperature. This methodology treats the CPU as a single node and monitors its overall temperature (computed from the total power) at macro level for thermal and energy management. Though simple, the hot spots of the chip, which are more critical in thermal management, cannot be observed. In Lee and Skadron [2005], the approach is improved through using HotSpot to compute the temperature for every FU in order to capture hot spots at runtime. However, significant overhead, both in performance ($\sim$20% slowdown to the monitored program) and temperature (increased by 13.65°C at the integer register file), was encountered because HotSpot adopts an accurate but inefficient integral-based numerical method to solve its thermal model in ordinary differential equations (ODEs). We

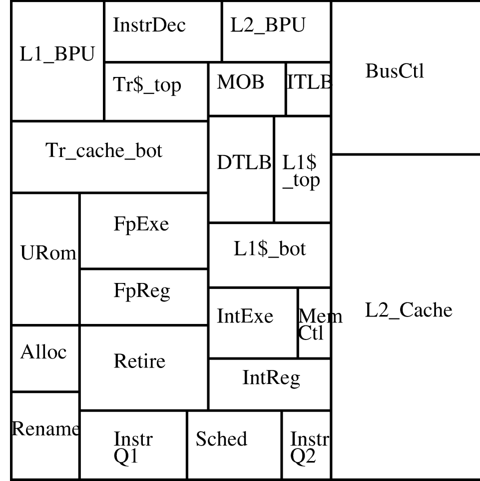

Fig. 1. P4 Northwood floorplan.

will leverage the formal *model order reduction* procedure that is used to transform high-dimensional linear ODE systems, such as an RC thermal circuit, into low-dimensional subspace for high efficiency in time and space [Celo et al. 2005; Li et al. 2005a; Liu et al. 2005; Rudnyi and Korvink 2005; Tsai and Kang 2000; Wang and Chen 2004]. Since our thermal model is relatively low-dimensional already, we perform a better and more stable procedure than the prior works in our runtime STS framework to make the temperature calculation accurate and nonintrusive at runtime.

3. SOFTWARE THERMAL SENSOR

Our STS has two main components: a power estimator and a temperature calculator. Both operate at the granularity of the FU. The power estimator periodically queries a set of performance counters online, and converts them into the average power consumption over the last time interval for each FU. The temperature calculator then uses the power to compute the temperature at each FU using the thermal model precalibrated according to the processor package, including the heat sink. It is possible to track power and temperature at finer granularity at the cost of runtime performance overhead. Such granularity is constrained by the limitation/capability of the hardware event-tracing counters only.

3.1 Power Measurement

Overview. We use a Pentium 4 Northwood core as our target processor. The FUs are extracted from its floorplan as shown in Figure 1 (adapted from chiparchitect.com [Floorplan 2007]). There are 22 FUs in total. The processor's total power during an execution of a program can be expressed as

$$\sum_{i=1}^{22} (Act_i \times \bar{P}_{i-dynamic} + P_{i-leakage}(T)) + P_{idle} = P_{total}, \quad (1)$$

where Act_i is the activity factor, namely, the number of times FU_i is used during a unit time, obtained through the performance counters. $\bar{P}_{i-dynamic}$ is the *average per-access* power of FU_i, including the dynamic power and a portion of its clock power (gated during idle time). The Pentium processor incorporates aggressive clock gating. When there is no workload, the processor gates off a large number of components to save clock power. When an FU is accessed, both its gated clock power portion and its dynamic power should be charged. For the nonclock gated portion, we merge these into P_{idle}, which represents the overall processor power in idle state. We treat $\bar{P}_i$ as the average (rather than the maximum) power as in Isci and Martonosi [2003], since an FU's power is proportional to the amount of signal toggling at runtime, which is impractical to trace. Therefore, we seek a statistical approach to finding the typical average power for each FU. $P_{i-leakage}(T)$ is the leakage power of FU_i. It is a function of the temperature T. Finally, P_{total} is the total processor power. Both this and P_{idle} can be measured from the CPU power lines externally. We will not explain how the performance counters are configured to capture the activities of each FU, as we followed the approach in Isci and Martonosi [2003]. The experimental setup for measuring the total power and logging the counters is also the same.

Solving FU power. Once we can attain all the Act_is and P_{total}, multiple instances of Eq. (1) can be created through running different programs so that a system of equations can be established where $P_{i-dynamic}s$ and $P_{i-leakage}(T)s$ are unknowns and the rest knowns. At the current stage, our main focus is to find the $P_{i-dynamic}s$, since temperatures are not available yet. We used the on-die thermal sensor readings, available in Pentium 4, as the estimation of T. However, this is only a single-point temperature of one FU. We will come back to revise this when we are able to calculate the *Ts* for all FUs. In prior work, the $P_{i-dynamic}$s are obtained empirically with experiments and fitting on microbenchmarks. We take a more mathematical approach to solving for $P_{i-dynamic}$s from the system of linear Eq. (1), established through executing a set of uncorrelated microbenchmarks so that the equations are all independent. The microbenchmarks are developed either in C or $\times$ 86 assembly language, and exercise different FU sets.

Our method works well for our microbenchmarks, as the measured and computed total power are very close (see Figure 2). The total power for each program is steady, so we show only a single value in the figure. However, when we use the $P_{i-dynamic}$s for significantly more complex programs such as the SPEC2K benchmarks, we observed large deviations. Figure 3 shows such an instance. In this program, the computed total power is less than the measured total power, indicating that some $P_{i-dynamic}$s are overly underestimated and need to be refined.

Reasons for inaccuracy. One reason for the deviation is because of the rough estimation of the leakage power, which can be corrected later when temperatures are available. Another important reason is because Eq. (1) has not yet covered all possible scenarios of power consumption. For example, we used the "add" instruction in a microbenchmark to exercise the integer ALU. However, the ALU has other arithmetic and logic operations such as "sub", "shift", etc., as

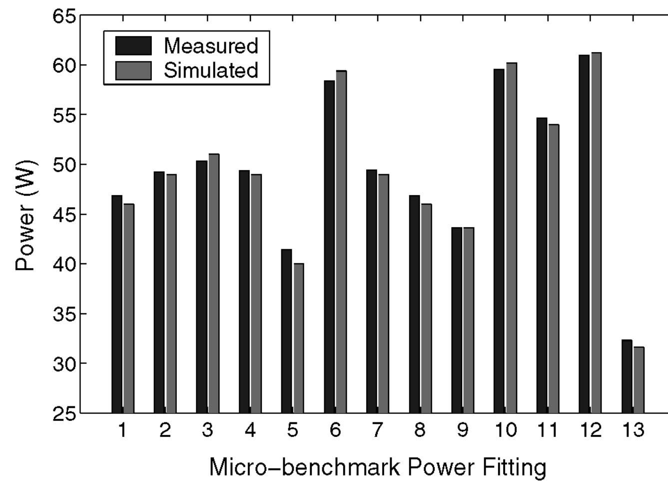

Fig. 2. The computed and measured power for microbenchmarks match very well.

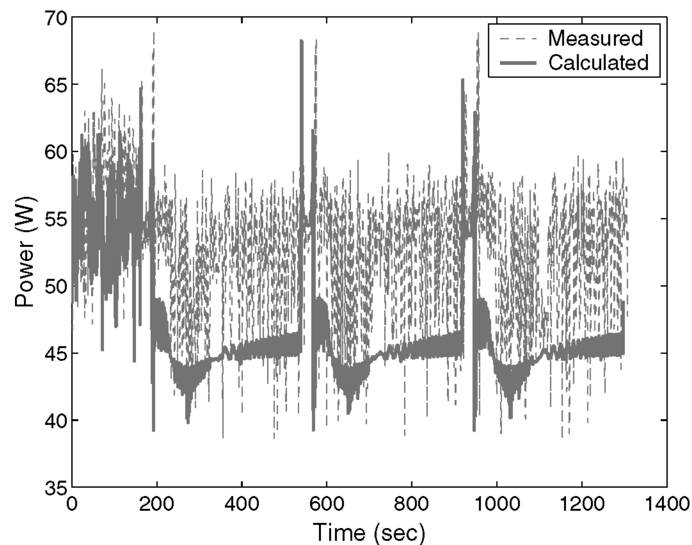

Fig. 3. The inaccuracy of using the power results from the microbenchmarks on a SPEC2K program `vpr`.

well. These may consume different powers than the “add” operation. Similarly, different binary inputs produce different internal switchings, which results in varying power consumption. Moreover, different amounts of bitline toggling in register files and caches generate different power dissipation levels as well. Hence, to cover all cases in microbenchmarks is not practical, since every benchmark is written by hand. Instead, it is more efficient to adopt a comprehensive set of benchmarks that are sophisticated enough to show vastly different cases. We choose the SPEC2K benchmark suite for this purpose.

Solution. We collected the counter and total power traces for the SPEC2K benchmarks. Unlike the microbenchmarks in which the traces are steady, the traces for SPEC2K oscillate drastically due to great variations in program behavior. It is inappropriate to sample random points from counter and power traces to form a new system of equations of the form of Eq. (1), as a single point in the power trace might carry some noise due to the measurement. Fortunately, recent studies on *program phase* behavior [Sherwood et al. 2002; Isci and Martonosi 2006] indicate that we can form our equations for phases instead of points. Since each phase represents a different power characteristic, our goal here is to cover as many phases as possible in order to establish a more complete set of equations for solving the FU power.

Finding phases from counter and power traces can be done using the *K-means* algorithm [Sherwood et al. 2002]. Essentially, it clusters the points into groups, with small intergroup distance and large intragroup distance for the points. Each group is one phase, with a centroid representing the phase. The two key components here are the *distance function* and the value of K, namely, *the number of clusters*. In order to consider counters and powers together, we form them into vectors such that each vector represents the counters and powers at the same time. Hence, the distance function is defined for the vectors and each cluster consists of vectors that are close to each other.

The distance function is typically the Manhattan function, that is, $Manhattan(\overrightarrow{X}, \overrightarrow{Y}) = \sum_{i=1}^{n} |x_i - y_i|$, where $\overrightarrow{X}$ and $\overrightarrow{Y}$ are vectors and n is their dimension. However, the Manhattan distance alone is not sufficient, since we consider the *correlation* between vectors as well. For example, if the first components of two vectors are the same, but the second components have opposite differences from the first components, even with the same distance, we do not consider them similar. The correlation of two vectors is statistically defined by the "correlation coefficient" r as

$$\begin{cases} r(\overrightarrow{X}, \overrightarrow{Y}) = \frac{cov(a,b)}{\sqrt{cov(a,a)cov(b,b)}} \\ cov(\overrightarrow{X}, \overrightarrow{Y}) = n \sum_{i=1}^{n} x_i \times y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i. \end{cases}$$

Here, r is in $[-1, 1]$. When $|r|$ approaches 1, the correlation is strong (i.e., linear) and when $|r|$ approaches 0, the correlation is weak. We now define our distance function to be the similarity between the two vectors.

$$Similarity(\overrightarrow{X}, \overrightarrow{Y}) = r(\overrightarrow{X}, \overrightarrow{Y}) \times (1 - Manhattan'(\overrightarrow{X}, \overrightarrow{Y})),$$

where $Manhattan'(\overrightarrow{X}, \overrightarrow{Y})$ is $Manhattan(\overrightarrow{X}, \overrightarrow{Y})/n$, to project it onto [0,1]. Hence, when two vectors are identical, their similarity value achieves the maximum, that is, 1. Using the similarity metric, we are able find a good value for K. To see this, we varied it and measured the minimal $Similarity(x, \varphi_j)$ (specifically, φ_j is the centroid vector of the jth cluster) for all $j = 1 \cdots K$. The result for the benchmark `gzip` is shown in Figure 4.

From our definition of similarity, the larger the K, the better the clustering and the larger the similarity metric. This is because when K is large, the points within a cluster are closer to each other, hence more similar to the centroid. As we see from Figure 4, after $K = 7$, the similarity is higher than 0.9, and the

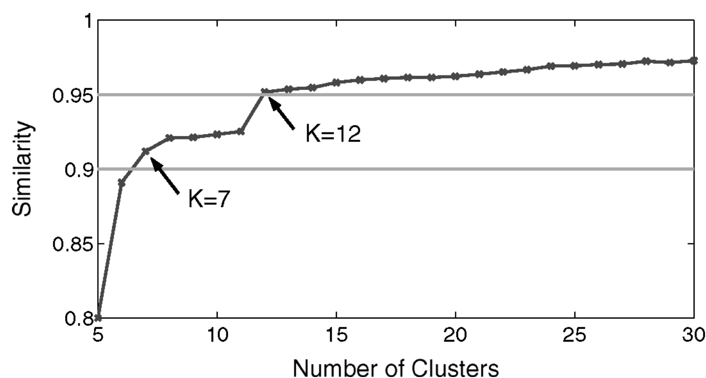

Fig. 4. The variations in similarity with K.

first time it exceeds 0.95 is when $K = 12$. However, the similarity does not grow linearly with K. After $K = 12$, the increase levels off. Therefore, we choose this value as our K, since it is sufficient. The Ks for other benchmarks can be derived similarly.

Once we find a good number of phases for the counter and power traces for all the programs, we form an *overdetermined linear system* as in Eq. (1) where the number of equations is more than the number of unknowns. In each phase, the relation between counters and powers is very similar to the microbenchmarks; their variation is very small. Therefore, the counter and power values can be obtained by averaging the points within a phase. In fact, this is the equation for the centroid of the cluster. We use the least-squares linear regression method in Matlab to solve the established overdetermined system of equations. For a linear regression problem, the greater the number of equations, the better the fit. Through experimenting with the SPEC2k benchmarks, we established a large number of equations that cover as many possible scenarios of power consumption as we observed. Also, it requires a good solution range to find the best fit. We use the solutions obtained from the microbenchmarks to create proper ranges. Such a method turns out to be quite effective. We will present the results of power estimation after providing detailed thermal distributions and better estimations for leakage power.

3.2 From Power to Temperature

Thermal model from HotSpot. At the architecture level, the thermal behavior of a microprocessor is modeled as an equivalent RC circuit, where the R is the thermal resistor and C the thermal capacitor, and both are dependent on the silicon parameters and manufacturing process. The power generated by a thermal node (e.g., a functional block) is modeled as the current flowing through a thermal resistor, and the temperature difference between the two nodes is modeled as the voltage difference between the two nodes. A simplified circuit for one node is illustrated in Figure 5. The lateral resistors ($R_{L_{1-4}}$) are the thermal resistor between the node and its neighboring nodes. When such an RC circuit is built for all blocks in the processor (assuming one functional block

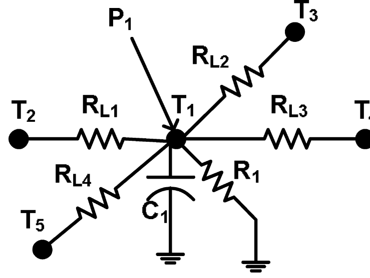

Fig. 5. A simplified thermal circuit for the center node T_1. Here $R_{L_{1-4}}$s are lateral thermal resistances, and R_1 and C_1 are the effective vertical thermal resistance and capacitance, respectively, between T_1 and the isothermal point. P_1 is the power generated by T_1, and the T_is are temperatures at neighboring nodes.

represents one node), the temperatures for all blocks can be solved from the resulting system of ODEs representing the circuit.

Specifically, for the simple thermal RC circuit in Figure 5, the temperature T_1 can be calculated by

$$\frac{T_1 - T_2}{R_{L1}} + \frac{T_1 - T_3}{R_{L2}} + \frac{T_1 - T_4}{R_{L3}} + \frac{T_1 - T_5}{R_{L4}} + \frac{T_1}{R_1} + C_1 \frac{dT_1}{dt} = P_1,$$

where the first five terms on the left are the heat flow from the center node T_1 to all neighboring nodes T_{2-5}, the sixth term is the heat flow from the center node to the ground, and the P_1 on the righthand-side is the total heat flow flowing through T_1. This equation is established by the Kirchhoff current law. If we write such an ODE for every node in the circuit and rearrange the terms, we obtain the matrix form of the resulting ODE system.

$$\mathbf{G}\overrightarrow{T} + \mathbf{C}\frac{d\overrightarrow{T}}{dt} = \overrightarrow{P}, \tag{2}$$

where $\mathbf{G}$ is a sparse coefficient matrix containing the conductances, $\overrightarrow{T}$ a vector of the temperature at all nodes, $\mathbf{C}$ a diagonal matrix for capacitors, and $\overrightarrow{P}$ a vector of the power injected into each node. Solving this ODE system for T with input $\overrightarrow{P}$ gives the temperatures for all nodes. The $\overrightarrow{P}$ is obtained online through the aforementioned counterbased scheme.

An efficient and stable numerical solver. The ODE of Eq. (2) can be solved through traditional numerical integration-based methods as in SPICE and HotSpot [Skadron et al. 2003]. Though accurate, such methods cannot be used at runtime due to their inefficiency. In our approach, we adopt the concept of *moment matching* (MM) in the frequency domain to compute the transient thermal response. MM methods exploit the fact that system behaviors are typically dominated by a few system states (i.e., their corresponding poles). These dominant states can be found by performing a moment matching of the input-output transfer functions of the system. Moment matching was proposed originally for reducing the system models (via so-called model order reduction (MOR)) [Pillage and Rohrer 1990]. Currently, Krylov subspace-based MOR methods are the most efficient model order reduction techniques because of their improved

numerical stability and moment matching connections of the reduced models [Antoulas and Sorensen 2001]. We notice that MOR techniques have been used for solving large finite element discretized thermal circuits [Celo et al. 2005; Li et al. 2005a; Liu et al. 2005; Rudnyi and Korvink 2005; Tsai and Kang 2000; Wang and Chen 2004].

Compared with previous models, our thermal model is relatively small in terms of node size and thus, our goal in applying MM is to meet the speed requirement for runtime execution. In fact, we applied MM to compute the transient responses directly instead of computing reduced models. However, when experienced with explicit MM methods like asymptotic waveform evaluation (AWE) [Pillage and Rohrer 1990], we often incur unstable (i.e., do not converge) solutions due to numerical problems and our system parameter requirement. To mitigate this problem, we apply a Krylov subspace projection-based method to compute the system poles. Our approach is close to Li et al. [2005a], but we further improve the previous approach by using more numerically stable pole searching methods (discussed next).

Since the standard procedure of MM can be found in the literature, we give only a sketch of this method with the description of our improvement in this article.

The MM Procedure:

(1) From Eq. (2), compute q *moments* $\overrightarrow{m}_i$ as

$$\begin{aligned}\overrightarrow{m}_0 &= \mathbf{G}^{-1}\overrightarrow{P}\\ \overrightarrow{m}_1 &= -\mathbf{G}^{-1}\mathbf{C}(\overrightarrow{m}_0 - T_0)\\ \overrightarrow{m}_2 &= -\mathbf{G}^{-1}\mathbf{C}\overrightarrow{m}_1\\ &\vdots\\ \overrightarrow{m}_{q-1} &= -\mathbf{G}^{-1}\mathbf{C}\overrightarrow{m}_{q-2},\end{aligned} \tag{3}$$

where T_0 is the initial temperature when STS is started.

(2) Use the modified Gram-Schmidt with double orthonormalization algorithm [Arnoldi 1951] to compute an $N \times q$ projection matrix V such that columns in V are unit vectors and mutually orthogonal, that is, $v_i^T v_j = 0, i \neq j$.

(3) Reduce the original $\mathbf{G}$ and $\mathbf{C}$ in Eq. (2) to two $q \times q$ matrices by the *congruence transformation*: $\hat{\mathbf{G}} = V^T\mathbf{G}V$, $\hat{\mathbf{C}} = V^T\mathbf{C}V$.

(4) Obtain the system's dominant *poles* p_i by finding the eigenvalues $\lambda_1, \cdots, \lambda_q$ of $\hat{\mathbf{G}}^{-1}\hat{\mathbf{C}}$, namely, $p_i = -\frac{1}{\lambda_i},\ \ i = 1 \cdots q$.

(5) Compute *residues* $k_{i,x}$s for every node x from

$$\begin{aligned}m_{1,x} &= -\left(\frac{k_{1,x}}{p_1} + \frac{k_{2,x}}{p_2} + \cdots + \frac{k_{q,x}}{p_q}\right)\\ m_{2,x} &= -\left(\frac{k_{1,x}}{{p_1}^2} + \frac{k_{2,x}}{{p_2}^2} + \cdots + \frac{k_{q,x}}{{p_q}^2}\right)\\ &\vdots\\ m_{q,x} &= -\left(\frac{k_{1,x}}{{p_1}^q} + \frac{k_{2,x}}{{p_2}^q} + \cdots + \frac{k_{q,x}}{{p_q}^q}\right),\end{aligned} \tag{4}$$

where $m_{i,x}$s are the xth component of $\overrightarrow{m}_i$.

Table I. Time Complexity of MM at Runtime

Equation	FP Multiplications	FP Additions/Subtractions
(3)	$N \times q$	$N \times q - N$
(4)	$N \times q^2$	$N \times q \times (q-1)$
(5)	Nq	$N(q-1)$

N is the number of nodes, q the number of poles.

(6) Plug in all the residues, poles, and the first moment $\vec{m}_0$ into equation

$$T_x(t) = m_{0,x} + \sum_{i=0}^{q-1} k_{i,x} e^{p_{i,x} t}, \quad T_x(t_0) = T_{0,x} \tag{5}$$

to obtain the asymptotic temperature function $\vec{T}_x(t)$ for every node x.

When the power $\vec{P}$ for a time window is known, $\vec{T}(t)$ can compute the temperature at time t in that window for all nodes in the thermal model. As we can see, the poles are in the exponents of $\vec{T}(t)$. Therefore, their accuracy is critical to the quality of the temperature. The value q determines how many terms are included in Eq. (5). The higher the q, the better the precision. An unstable solution may generate positive poles which lead to exponential growth of the temperature in time. Steps (3) and (4) guarantee that the reduced system $\hat{\mathbf{G}}^{-1}\hat{\mathbf{C}}$ produces negative poles and hence a stable solution. We found that the first few dominant poles found in Li et al. [2005a] may not be very accurate due to their large magnitudes through using a simple orthogonalization process (e.g., the Gram-Schmidt method). As a result, in our Step 2, we apply the Arnoldi method using a modified Gram-Schmidt with double orthonormalization to further improve the numerical stability of finding more poles. This helps us in searching for a good number for q, as we require good precision in temperature and low complexity in time.

Time and space complexity. Not every step in the MM procedure needs to be performed at runtime. In particular, poles reflect the characteristics of a thermal circuit and do not change with the power inputs. Therefore, we need only compute them once when the thermal model is fixed. For the moments and residues, some of the operations can also be prepared beforehand, greatly reducing the overhead at runtime.

Specifically, the quantities that can be precomputed are: e^{p_i} in Eq. (5), $\mathbf{G}^{-1}$ in Eq. (3), and inverses of the coefficient matrices in Eq. (4). Let N be the number of nodes in the thermal circuit, Table I summarizes the number of operations for getting the moments from Eq. (3), the residues from Eq. (4), and the temperature from Eq. (5). In our STS, $N = 82$ (adopted from HotSpot). We experimented with the number of poles in the system and found that when $q = 6$, MM gives high accuracy. More poles give better accuracy at the cost of overly high space and time overhead.

The MM procedure needs some memory space to remember the precomputed and temporary vectors and matrices. The total space required is summarized in Table II. The column labeled "Temporary" is for all the temporaries computed for runtime computation. Note that the $N \times q$ residues and moments can

Table II. Space Complexity of MM

Equation	Precomputed	Temporary
(3)	$\frac{1}{2}N$ (for $\mathbf{G}^{-1}$) N (for $\mathbf{C}$)	$N \times q$(moments) FU (average power)
(4)	$q \times q$ (coefficient matrix)	$N \times q$ (residues)
(5)	q (for e^{p_i})	N (temperature)

$FU = 22$ is the number of function units.

share a common space, since the moments are not needed once the residues are computed. Also, C is diagonal and thus can be stored as an N-dimension vector.

Speedup and accuracy. Our MM method has great advantages in time over the nonadaptive fourth-order Runge-Kutta (RK) method used in HotSpot for computing the temperatures. This is because we compute only a single *transfer function*—Eq. (5)—in every time window. The temperature can then be computed by plugging t at that window into our transfer function. Therefore, there is only a *constant* overhead in each time window. However, in the RK method, the time window must be divided into many steps to achieve the required accuracy, and a temperature at any time must be obtained after all previous temperatures for each step are calculated. It is a step-by-step iterative method whose computation overhead increases linearly with time.

To see the efficiency of MM quantitatively, we collected real power traces for 200 points with a 0.01s sampling rate. The powers are for the integer register file produced by our performance counters. We let both algorithms compute the temperature using the same thermal model and the same power input traces. The temperatures were computed on every power value. We timed their execution durations on a real machine. Both algorithms are optimized by the Intel C++ Compiler 9.0 [Intel 2007a] with the Math Kernel Library 8.0 [Intel 2007b] for fast matrix multiplication. A speedup of $3\times$ for the algorithms themselves was observed with the optimization. The execution time includes disk readings for the traces, so there are slight differences among different programs. The average speedup we observed for MM with respect to RK was $\sim$**17000**. For longer power traces with the same 0.01s sampling rate, the speedup is the same if disk reading time is excluded. If the sampling rate (or the temperature calculation interval) is increased, the speedup will also increase, as the overhead for MM is always constant irrespective of the interval length, while for RK the overhead is linear to the length of time interval. When the MM method is embedded into our STS, there are other runtime overheads such as collecting the performance counters, computing powers, and the interrupt itself. The overall overhead will be discussed in Section 7.

The MM method also achieves excellent accuracy compared to the RK method. The RK method itself is very accurate and comparable to SPICE, since both are integration-based solvers for circuits. For the preceding experiments, we also measured the deviations in calculated temperatures between MM and RK. The results are shown in Figure 6. As can be seen, the differences (in absolute value) are always within 0.1°C with an average of 0.03°C, and keep in mind that this is the temperature for the integer register file, which is typically

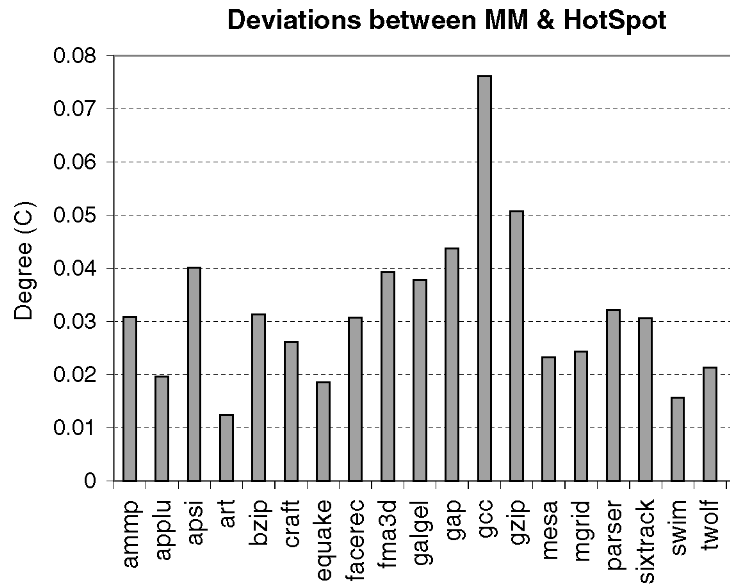

Fig. 6. The deviations between MM and Runge-Kutta (used in HotSpot).

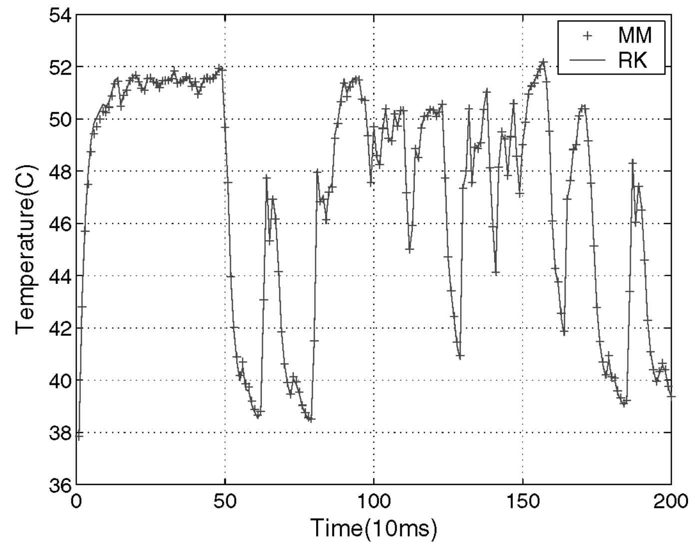

Fig. 7. The temperature curves obtained from MM and RK for gcc at the integer register file unit.

the hottest [Skadron et al. 2003]. We also show in Figure 7 the temperature curves for gcc, which has the largest average deviation in Figure 6.

3.3 Leakage Modeling

Now that we can compute the temperatures for every FU, the next critical step is to plug them back into Eq. (1) to form a closed loop with better estimations

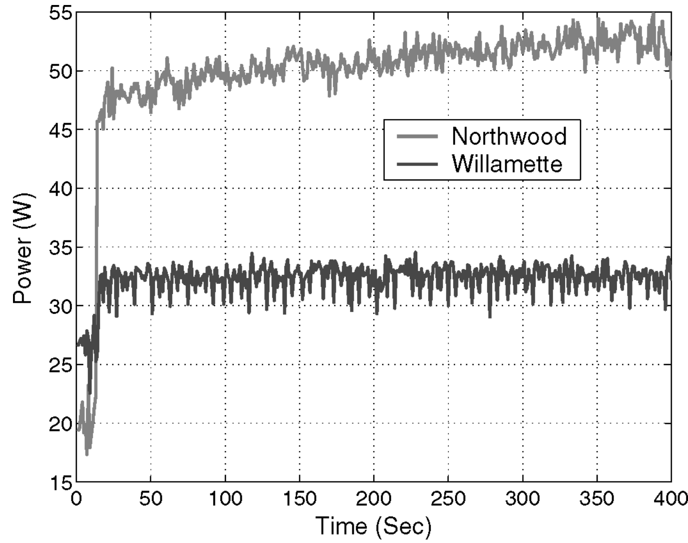

Fig. 8. The total power measured for Willamette and Northwood running the same benchmark. Northwood shows an increase in leakage power.

of the leakage power. Leakage power becomes dominant as technology size shrinks. Particularly, it grows exponentially with the temperature. The differences are significant in thermal results, both with and without leakage counting, as shown in Huang et al. [2005]. We also observed noticeable differences in the measured total powers for P4 Willamette (1.6 GHz, 0.18 μ) and Northwood (2.8 GHz, 0.13 μ) running the same workload (see Figure 8). The architectures of the two cores are the same, except that Northwood doubles the L2 cache size. The program is a simple infinite loop on a dummy integer multiply instruction. The total power in Northwood increases, while this program indeed produces constant activity and dynamic power, as with Willamette. This is because the leakage in Northwood is more evident due to its smaller technology size and higher average working temperature.

Leakage power includes both gate and subthreshold leakages. It can be expressed as [He et al. 2004; Li et al. 2006]

$$\begin{cases} P_{leakage} = Gate\ Count \times I_{leakage} \times V_{dd} \\ I_{leakage} = A \times T^2 \times e^{-B/T} + C \times e^{r1 \times V_{dd} + r2}, \end{cases} \tag{6}$$

where *Gate Count* is the number of transistors in a circuit. It can be estimated to be proportional to the FU's chip area. $I_{leakage}$ is the leakage current, T the temperature, and $A, B, C, r1, r2$ are constants. The first term of $I_{leakage}$ denotes the subthreshold leakage that grows exponentially with temperature. The second term denotes the gate leakage, which is insensitive to temperature changes and can be included in the chip's idle power. Therefore we focus on the subthreshold leakage modeling. To obtain A and B, we performed SPICE simulation using the PTM 0.13μ technology parameters [ASU 2007], matching the Northwood technology size. We varied T in Kelvin and plotted the leakage current curve

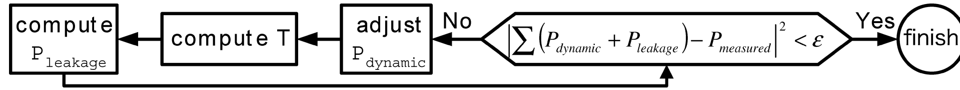

Fig. 9. Iterative power estimation considering the leakage.

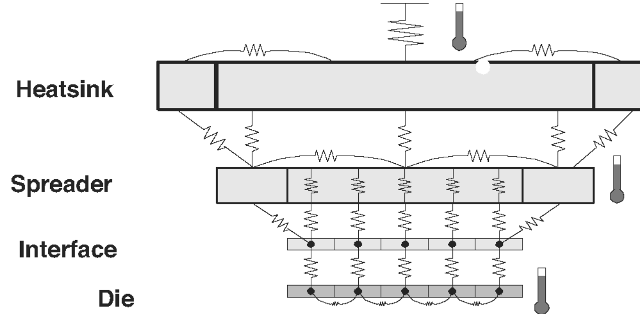

Fig. 10. The RC network of our thermal model.

for a transistor. A and B are decided by fitting the curve from Eq. (6) with the resulting curve from SPICE. Through this, we found that $A = 550.54$ and $B = 5082$.

Equation (1) can be revised to include the newly computed leakage power. It can be solved again for better $P_{i-dynamic}$'s. This procedure should iterate several times, until the difference between the measured and computed total power is small, as depicted in Figure 9. Moreover, this procedure should be repeated later, as the temperature results also depend on the thermal model's parameters. When the temperature is adjusted, the leakage power is changed as well. In the next section, we describe how we calibrate the model parameters from real power measurements, sensor readings, and thermal images, followed by the final power estimation results.

4. CALIBRATION AND VALIDATION

Recall that in the simplified RC thermal circuit shown in Figure 5, all the resistances and capacitances were assumed known a priori. This is difficult, unfortunately. In fact, we experienced that a small variation in certain R and C values can lead to a significant deviation in temperature. Figure 10 shows the RC network extracted from our thermal model, which is adapted from Skadron et al. [2003] and Huang et al. [2005, 2004]. Only the primary heat conduction path is considered, as it accounts for most of the heat removal from the die [Huang et al. 2005]. There are five layers in the model, representing the die, thermal interface material, two sides of the heat spreader (one up, one down), and the heat sink. The die is discretized into 22 nodes, each corresponding to one FU. Upper layers have fewer nodes than bottom layers, since the heat is more spread out as it is removed.

In general, every node has lateral resistors with its neighboring nodes on the same layer, and a vertical resistor with its immediate upper or lower layer. Every node also has a capacitor with respect to the ground. An R value determines

the temperature drop/rise between two neighboring nodes, either laterally or vertically. A C value determines how quickly the node temperature changes with the node's power. Therefore, for example, smaller *vertical* Rs represent a package with better heat conductivity and therefore the chip temperature will be cooler. We used the RC values from the HotSpot tool initially. If the computed temperatures are not calibrated with a real setting, the results cannot be trusted. For example, for two models with slight differences in $R_{spreader\text{-}to\text{-}sink}$, let a cool program run on a model with a larger $R_{spreader\text{-}to\text{-}sink}$ and a hot program run on the other. The temperature observed from the heat sink might be the same, but internally one die is hotter than the other.

4.1 Real Measurements

As we can see, it is critical to calibrate first the vertical Rs in the model, since the heat dissipates rapidly in this direction and relatively slowly across a surface. This requires that we obtain true temperatures at different layers so that the temperature discrepancies among them are explicit. Apart from vertical Rs, the lateral Rs should also be calibrated. Initially they are set to be proportional to each FU's area, as in HotSpot. During our calibration their values are refined gradually. We used five real measurements: three point measures at three different layers, one surface measure at the heat spreader, and one measure in ambient air, for the calibration on the RC values.

1. *An on-chip thermal diode reading.* The Pentium 4 employs two on-chip sensors, one visible externally, but the other not [Intel 2002]. To access the visible sensor (in fact, a thermal diode), we used an Intel Desktop Board D865PERL that supports an SMSC EMC6D101 [SMSC 2007] off-chip device for power/temperature monitoring and fan control. This device converts on-chip diode readings into digital temperature and transmits it via the SMBus interface. We use a driver to read the SMSC internal temperature register with a maximum sampling rate of one temperature measurement per second. This includes the delay from the diode to the SMSC, temperature conversion inside the SMSC, and the device driver's latency. This rate is sufficient for the purpose of calibration.

2. *A thermometer reading on the heat spreader.* We attached a Meterman Type K thermocouple to the spreader on its edge corresponding to one side node in our model. Its readings are logged through a DMM at a rate of 50 values/second. The touch point is applied with thermal grease to ensure good heat conductivity.

3. *A thermometer reading on the heat sink.* We also attached a thermocouple to the heat sink. Since the sink bottom is $\sim$1–2°C higher than its top portion (close to the fan), we took several individual measurements and obtained the average temperature of the heat sink under the same workload.

4. *Thermal images taken for the heat spreader.* We used an infrared (IR) thermal camera "ThermaCAM SC2000" from the FLIR system to take a series of thermal photos for our Northwood heat spreader. In order to do so, the heat sink must be removed from the package, which will lead to a failsafe shutoff of the entire machine when the internal temperature is approximately 135°C

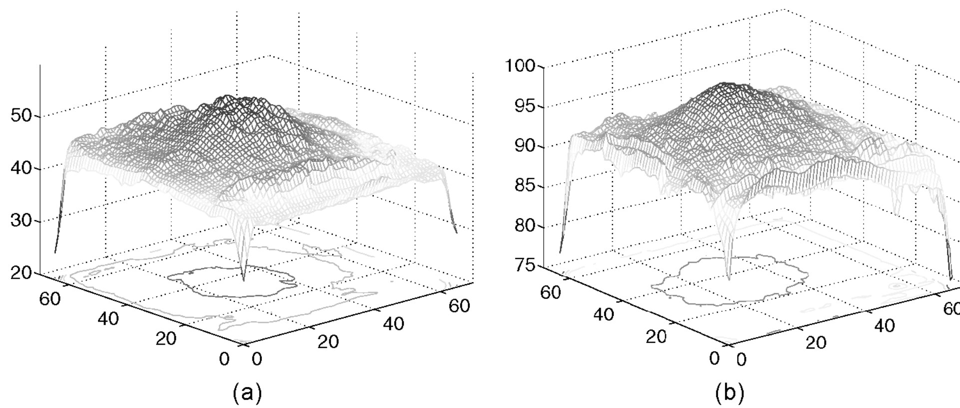

Fig. 11. Thermal images of the heat spreader for an FP microbenchmark taken when the heat sink has just been removed (left) and at the failsafe shutoff point (right). The bottom plane shows two temperature contours.

[Intel 2002]. This duration is typically 10 seconds after the heat sink is removed, whereas our IR camera takes one thermal picture per second. Therefore, we had roughly 10 pictures before the shutoff for each tested program. These images show the dramatic internal temperature upsurge until the shutoff point, after which the chip starts to cool down. Figure 11 shows the 3D plots of the thermal images for an FP microbenchmark, after the heat sink has just been removed (11(a)) and right at the shutoff point (11(b)). The latter obviously has a much higher thermal profile. The die is located in the center underneath the spreader. We can see that the central hot region amplifies with time due to the absence of the heat sink.

The thermal images are used to calibrate the lateral Rs and Cs for each node modeled on the spreader. When interpreting the images, we must determine the correct emissivity for the surface material of the spreader. In order to remove the noise (i.e., engraved chip model information) of the clean spreader, we applied a layer of thermal grease to make the surface material uniform. Afterwards, since we also have a thermocouple attached to one edge of the spreader, we adjusted the emissivity until the traces of the thermocouple readings and the thermal image readings at that point match well.

5. *A thermometer reading for the ambient air.* The ambient air temperature has a large impact on heat removal from the package. We found that during program execution, the air temperature can increase $\sim 10^{\circ}$C. Even when we execute programs in a closed-case machine, the chip temperature is much higher than an open-case machine for the same program binary and system setting. The former triggered duty cycle throttling (i.e., lowered the frequency) and slowed down the program. Therefore, considering the temperature changes in ambient air is also a critical factor in our thermal modeling.

Modeling the air flow and its temperature mathematically is currently beyond the scope of this article. It will be an interesting future work to explore. We took the approach of measuring the temperature changes around the heat sink and curve-fit them using two on-board sensors that are available on our

Table III. Parameters Calibrated for Our RC Model

Name	Meaning
R_convec	Thermal Res. between heatsink and ambient air
C_convec	Thermal Cap. between heatsink and ambient air
t_heatsink	Thickness of heat sink
t_spreader	Thickness of spreader (with thermal grease)
t_tim	Thickness of thermal interface material
C_factor	Fitting factor for global factor
R_SI	Unit thermal Res. for chip
R_CU	Unit thermal Res. for spreader
R_TIM	Unit thermal Res. for interface material
Heat_SI	Unit thermal Cap. for chip
Heat_CU	Unit thermal Cap. for spreader
Heat_TIM	Unit thermal Cap. for interface material

Intel motherboard. When such a function is established, the on-board sensor readings can be used to compute the ambient air temperature around the heat sink at runtime, which is then fed to the thermal model.

4.2 Calibration Method

After collecting the measured temperature traces, we proceed to calibrate the RC matrix in our model in order to match the simulation outputs with the real readings. Note that all traces must be first aligned in time, as they are logged with different delays. To narrow down the search space, we do not take all the RCs as variables. We assume that the relative ratio between FUs in same layer is correct, as suggested in Skadron et al. [2003], with the only exception in the heat spreader (for which the RC values can be calibrated from the thermal images). Other adjustable vertical RCs are those from air to heat sink, from heat sink to spreader (including the thermal grease), and from the spreader's lower side to the die.

Finally, we choose the 12 parameters listed in Table III to tune. Our objective is to minimize the squared error summed over all the programs we measured. This is defined as

$$error = \sum_{all\ programs} |T_{measured} - T_{simulated}(x_1, x_2, \dots, x_{12})|^2,$$

where $x_1, \dots, x_{12}$ are our parameters to be adjusted. This is a minimization problem that can be tackled by the *conjugate gradient* (CG) method [Stoer and Bulirsch 1991], which is an algorithm for finding the nearest local minimum of a function of n variables. It uses conjugate directions instead of the local gradient for going downhill. If the vicinity of the minimum has the shape of a long, narrow valley, the minimum is reached in far fewer steps than would be the case using the method of steepest descent.

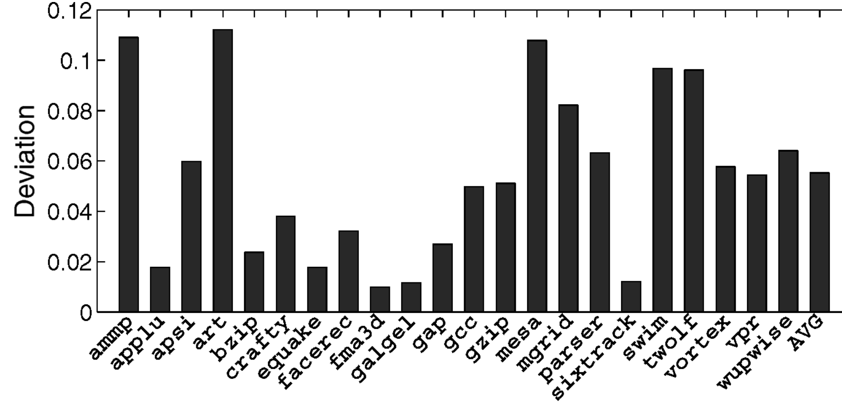

Fig. 12. Deviations between measured and calculated total power.

However, as CG gives only a local minimum, it may not be the global minimum for our objective function. We repeat the CG a number of times. Each time, we add a random offset to the computed result and start the next round. We then choose the lowest local minimum as our global minimum. For our thermal model with 12 variants, we let it iterate 50 times on average to obtain a solution. Note that at this time we need to go back to the power calibration loop as depicted in Figure 9, since the model has been calibrated and thus the computed temperatures will be different from before. Such an iteration needs to be carried until both the power and temperature results are within the requirements. We experienced that a couple of iterations are typically sufficient. Finally, the measured temperature traces are selected from a subset of our total benchmark set and the calibrated RC values are used to test against the complement set to see its accuracy. The results will be shown in Section 6.

5. POWER RESULTS

With the calibrated model and leakage power modeling, the estimation of the powers for FUs can be now finalized. We show these results in terms of "deviations" between measured and calculated total power (Figure 12) because the measured power may have noise from different sources. Since we used K-means to cluster the measured power when solving Eq. (1), we took the averages within each cluster to suppress the noise. The comparison is shown in Figure 12 where the deviation is defined as

$$Deviation = \frac{\sqrt{\sum_{i=1}^{i=n}(Measured_i - Computed_i)^2}}{\sqrt{\sum_{i=1}^{i=n}(Measured_i)^2}} \times 100\%.$$

As we can see, the deviations for these 22 benchmarks range from low ($<$ 2%) to medium ($\sim$5%) to high ($\sim$11%), with an average of 5.33% deviation. In Figure 13, we compare the traces for the measured and computed power for representative programs with low, medium, and high deviations. It can be seen that program `fma3d` has been improved significantly compared

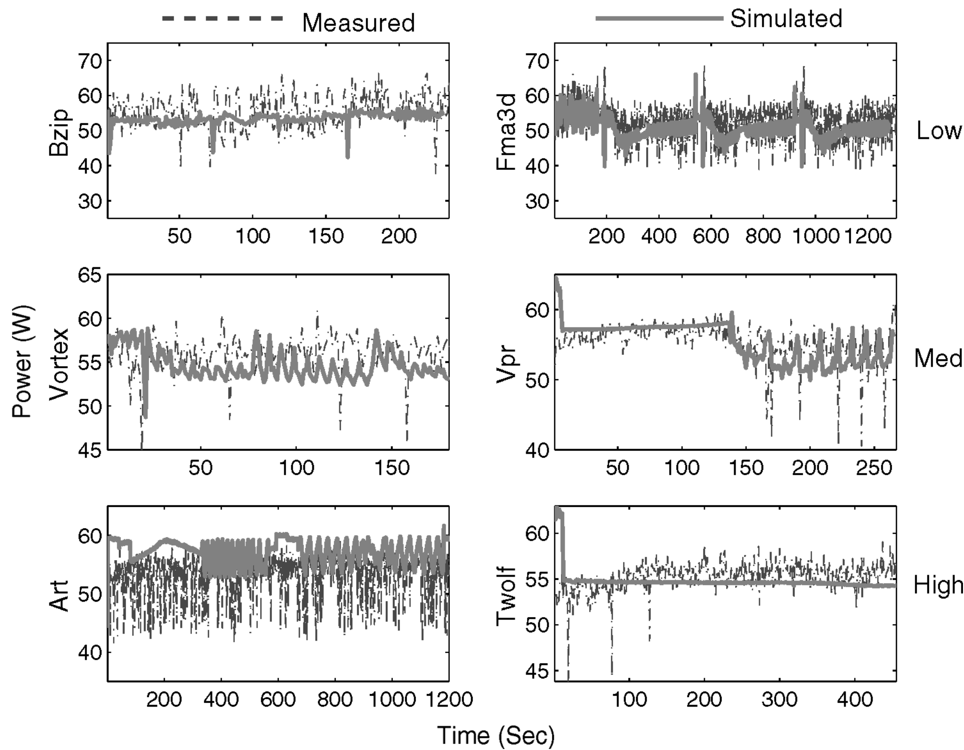

Fig. 13. Comparison of measured and calculated total power for low, medium, and high deviation rates.

to Figure 3, demonstrating the effectiveness of trace clustering, incorporating leakage power, and model calibration. However, for programs such as `art`, the deviation is still quite large. We found that its measured power oscillates drastically while that of the performance counters does not. Intuitively, the counters and power should be "consistent" to certain degree. This program is counter-intuitive and may indicate that there are some hardware events that are not captured by the performance counters due to their limitations. Hence, the power related to these missed events is charged to other FUs. This results in higher computed than measured power. Another program, called `mesa` (not shown), also has a higher computed power than the measured power curve. For this program, we observed the highest L2 cache accesses and misses, and noted that its instruction retire rate is the lowest. Therefore, we reason that there might be aggressive clock gating in the core, lowering the total power consumption. Those factors cannot be captured by the counters either. Overall, using performance-counter-based power metering can achieve good approximations, as the average deviation is modest.

6. THERMAL PROFILES AND OBSERVATIONS

To see how well our temperature calculation performs, we let it run for an extended duration. We ran the 22 SPEC2K benchmarks back-to-back with a

3 minute gap between each (so that their boundaries are clear). Each program takes the reference input and repeats 3 times to heat up the processor sufficiently. The total span of this experiment is about 8 hours, during which the STS monitors the chip continuously on 1-second intervals. We let it run at a sparse interval because it collects and logs all online information, including the counters, powers, and temperatures for all the nodes in our thermal model. We are interested in the final results that show the temperature variation in each FU and other nodes in our model for all the programs. In Figure 14, we plot the temperature profiles for all the benchmarks. We omit overlapping curves to make this figure easier to read. The temperatures for major FUs (listed on the right), on-die thermal diode readings, measured temperatures on the heat spreader, and ambient air are plotted in the figure. The average power of each program is also shown at the bottom of the figure.

First of all, one purpose of this experiment is to show that the errors arising from model abstraction, power estimation, and temperature calculations do not propagate over time. This is important as there are three levels of estimation in our STS: activity to counter, counter to power, and power to temperature. Error occurring in any level may lead to a diverging temperature after a certain time. The results show that the measured temperatures on-die and at the spreader match very well with our computed values (the computed spreader temperature is denoted as SimSp_t) throughout, demonstrating that our estimation temperatures are reliable. In addition, there are several observations we make from this figure:

(1) All curves for all programs have a "Γ" shape. This complies with the exponential property of temperature variation in time. In general, every program has a "stable" temperature (where the temperature levels) that is determined by its *average power consumption*. Higher average power corresponds to higher stable temperature, as can be seen from the figure. However, this statement should be revised to consider the ambient air temperature. The reason that some programs become stable and some increase in three continuous runs is because their ambient air temperatures are different. When the air is at a stable temperature, so is the die temperature (e.g., `art` and `swim`). When the ambient air keeps warming up due to a fast rise in chip temperature, the so-called "stable" temperature of the program is also elevated (e.g., `crafty`, `gcc`, `parser`, `wupwise`). However, the "stable" temperature will not keep increasing unbounded. It will still reach a stable value (higher) after a sufficient amount of time, just like the program `art`. We can view the ambient air as an additional layer of heat conductor with a large thermal capacitance. Such a phenomenon could not be observed by previous thermal models [Skadron et al. 2003], as the air was assumed to be at a constant room temperature.

(2) The temperature range on-die varies from 10°C (`art`) to nearly 20°C (`gzip`). Figure 15 shows the thermal map for the peak point in Figure 14. Here the integer register file is 80.5°C and the coolest FU, namely, bus control, is 59°C. Even for a single FU such as the DTLB and integer register file, the temperature can oscillate dramatically, with a magnitude of 12∼15°C

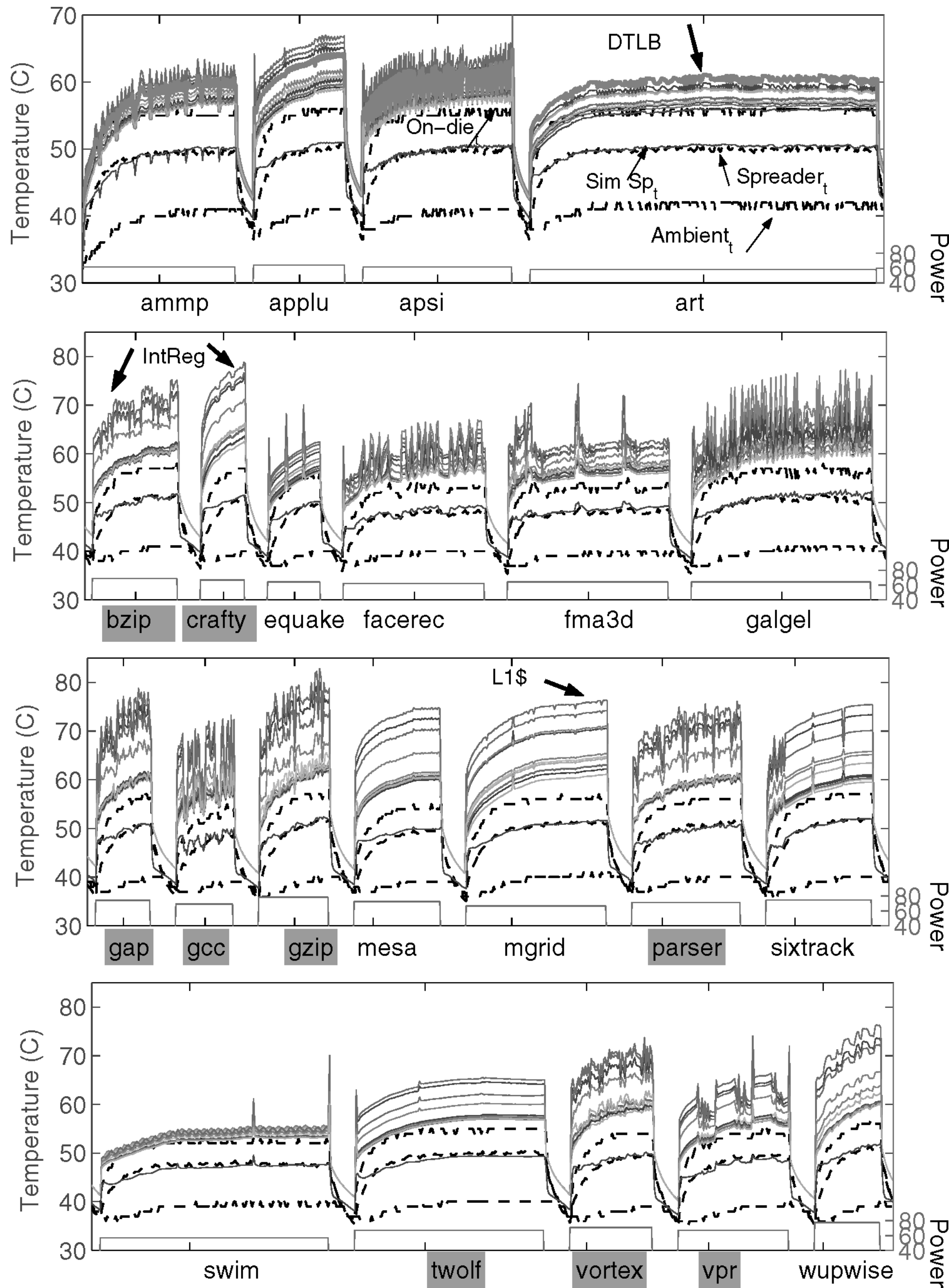

Fig. 14. Thermal profiles for continuous execution of 22 SPEC2K benchmarks for a duration of 8 hours. Each program is repeated 3 times to show its thermal behavior over time.

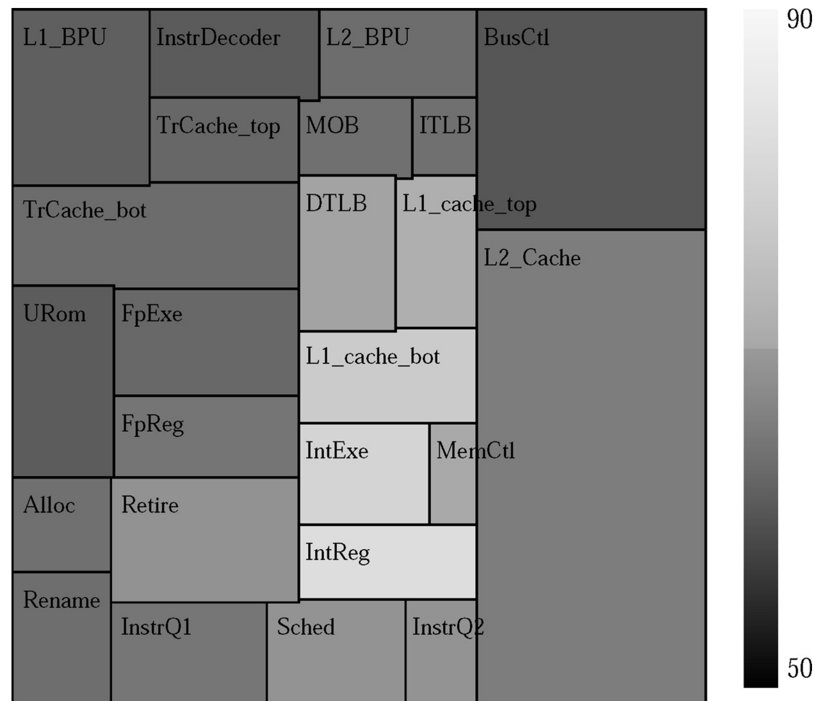

Fig. 15. The thermal map of gzip at the peak point.

within one second (galgel, gcc). Such a feature creates a great thermal cycling problem at high temperatures, which negatively impacts the reliability of the chip [Srinivasan et al. 2006].

(3) The three hottest FUs we observed were: integer register file (bzip, gap etc.), DTLB (art, facerec, galgel), and the L1 data cache mgrid. For the integer register file, our observation is consistent with other thermal modeles [Skadron et al. 2003] even for a different microarchitecture. This is probably due to the nature of typical written programs. For DTLB, we observed through performance counters that these programs exhibit unusually high load playbacks, namely, misspeculated loads that are reexecuted [Intel 2004]. This is an architecture-dependent thermal problem. The L1 cache can sometimes become the hottest, as with program mgrid. We observed that this program accesses the L1 cache nearly once every two cycles. It is the highest in L1 cache request density in all programs. This high access rate, together with the small L1 cache area and the floorplan (see Figure 1), cause L1 to be the hottest FU in this program. We feel that in general, the hot spots of a program are due to heavy accesses to the corresponding FUs, which lead to increased local power consumption and temperature.

(4) Some programs have a single hottest spot (e.g., mesa, etc.) while others have more than one (e.g., equake has two: integer register file and the L1 cache). Also, there are no distinct differences between FP and integer benchmarks

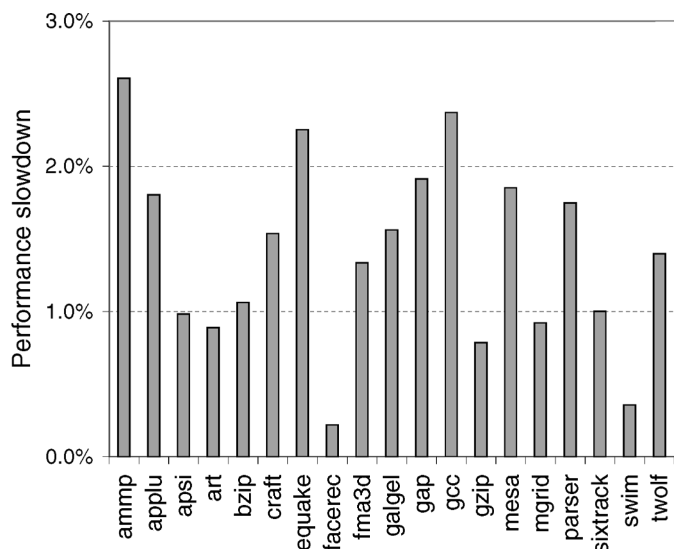

Fig. 16. Performance degradation due to STS.

(names are shaded in the figure), as most FP programs are also hot at the integer register file (e.g., `sixtrack`, `wupwise`). Moreover, the hot spots of one program may be the cool spots of another, for example, the DTLB is the hottest in `galgel` but cooler in `gcc`.

There are many more interesting observations that can be made through our profiles. We only name a few here due to space limitations. Also, some occasional spikes we saw in the figure are due to the switching between one run to the next (e.g., `fma3d`, `swim`). During this time, the instruction retire rate is extremely high and our script also does a recompilation of the program before a new run, introducing high on-chip activities.

7. STS RUNTIME OVERHEAD

Earlier we demonstrated that our MM temperature calculator, the core of the STS, is very efficient. The power estimator is simply the linear combination of the performance counters. These two components, when optimized by the Intel compiler, take $\sim$40 μs to execute in one time interval. Compared to that, reading the counters is very lightweight; only 18 assembly instructions in total. Such a small overhead allows STS to execute at fine time granularity if necessary. The remaining time is spent in the wrapping kernel module we embedded into the Linux kernel 2.6.9. The entire module consists of counter readings, power estimations, and temperature calculations. The STS itself takes some time for the context switching and loading precomputed matrices and vectors. To see its overall performance overhead with everything considered, we ran all the 22 SPEC2K benchmarks three times with reference input and measured the time taken both with and without the STS. The STS is invoked every 10 ms. The slowdowns with our STS are shown in Figure 16. All the programs incurred

less than 3% slowdown with an average of 1.5%. This is acceptable, considering that the temperature maps are computed 100 times per second.

8. SOURCES OF ERROR AND LIMITATIONS

We have demonstrated that our STS can provide runtime power and thermal profiles for various workloads to a trustable level. However, we do not recommend it for mission-critical temperature control such as detecting an emergency peak temperature. Rather, we consider this the responsibility of the hardware, since critical situations should be responded to immediately. Our STS, on the other hand, complements many commodity processors with the capability of showing thermal trends and the relative temperature among different FUs. Next, we will discuss possible error sources and the limitations of our STS, with suggestions on model enhancements.

To calibrate our results, the most important information we need is the on-die temperature values and their distribution. However, the only chip information available externally is the on-die thermal diode reading. Though useful to a certain extent, its location is not well documented and we found that it is not located close to the hottest spot. Since we used its readings to calibrate the chip temperature at one point, its location is thus important to us. A location at the L1 cache would give different results than a location in the branch prediction unit. We did an extensive search on the temperature curves produced by all FUs on a number of model settings, and found a curve (i.e., the bus control unit) with a shape closest to the trace of the diode reading. We therefore assumed that the diode is located at the bus control unit (see Figure 1). However, this is only a reasonable estimation and could affect the RC values of FU nodes in the thermal model if the diode is located differently. To overcome this, one method would be to derive the thermal model for a package without a heat spreader, such as the Pentium M (packed with only a thin cover). Using the thermal images of the slightly covered die could lead to much better precision of the on-die temperature variation.

Another source of error is the ambient air temperature. Our STS currently uses a function of two on-board sensor readings as a proxy to the air around the heat sink. This could lead to error, as the sensors are still away from the heat sink. Moreover, many heat sinks are installed with a fan that operates at different speeds. This can lead to a steep fall or slow rise in air temperature. With such a setting, the model for ambient air needs to consider the fan speed at runtime, which is feasible, as most fan speeds are controlled by an off-chip thermal diagnose device (e.g., the SMSC in our board).

The limitations of performance counters can lead to error in accounting for the power consumption. The Pentium 4 supports a total of 45 hardware events, but only 18 physical performance counters. This limits the types of events that a counter can capture. Therefore, it is possible that the total power is dissipated, but the counter missed some events, leading to inaccurate power reporting and temperature calculation. This is difficult to overcome unless more performance counters are implemented on chip.

Lastly, porting our STS to other systems is challenging at present. The power estimation and temperature calculation can be easily ported to a different

system. The most difficult part is the calibration, as thermal resistances and capacitances vary from package to package. Secondly, different heat sinks differently impact the die temperatures noticeably. The thickness of the thermal grease between the heat spreader and heat sink can make a difference of $2 \sim 3^{\circ}$C in the on-die diode readings. Also, different environments variously affect the ambient air around the heat sink. However, our experience is that a skilled person with proper experimental settings can carry the entire calibration in several days to one week (even though the learning curve might be steep). We could build a database of the thermal R and C parameters for different package components such as the heat sink, and interpolate new models based on existing data. We could also leverage in-chassis thermometers and on-chip thermal sensors to perform live calibration and adaptation. Our goal for the future is to automate all procedures to make our STS easy to port.

9. POTENTIAL APPLICATIONS

The applications of our STS can be manyfold. First, STS complements hardware on-chip sensors by providing comprehensive information about the processor at runtime. Hardware sensors can only report a few discrete spot temperatures, while the STS can report both temperature and power information for all functional units on-die. It can also be further discretized to model interesting functional units at finer granularity, so long as the total overhead is kept low. Second, software can make full use of STS to perform online power and temperature monitoring. This can lead to thermal-aware dynamic compilation designs or thermal-aware OS task scheduling mechanisms. Both will benefit from the proactive thermal estimation capability provided by the STS.

10. SUMMARY

In this article, we present a methodology for estimating the runtime power consumption and temperature distribution, as well as for calibrating the thermal model for a complex high-performance Pentium 4 Northwood core. We developed a software thermal sensor based on our methodology for a Linux system, and demonstrated a comprehensive set of thermal profiles for 22 SPEC2K benchmarks. Our temperature calculation method is as accurate as integration-based circuit solver such as SPICE, yet orders of magnitude faster. Our STS introduces only $<3\%$ performance overhead to the workload under monitoring. The overall accuracy of the STS is validated against measured temperatures of our processor package.

REFERENCES

ANTOULAS, A. C. AND SORENSEN, D. C. 2001. Approximation of large-scale dynamical systems: An overview. *Int. J. Appl. Math. Comput. Sci. 11*, 5, 1093–1121.

ARNOLDI, W. 1951. The principle of minimized iteration in the solution of the matrix eigenvalue problem. *Quat. Appl. Math.*, 17–29.

ASU. 2007. Predictive technology model. http://www.eas.asu.edu/ptm/.

BELLOSA, F. 2000. The benefits of event-driven energy accounting in power-sensitive systems. In *Proceedings of the 9th ACM SIGOPS European Workshop*.

BELLOSA, F., WEISSEL, A., WAITZ, M., AND KELLNER, S. 2003. Event-Driven energy accounting for dynamic thermal management. In *Proceedings of the Workshop on COLP*.

BROOKS, D. AND MARTONOSI, M. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, 171–180.

CELO, D., GUO, X., GUNUPUDI, P. K., KHAZAKA, R., WALKEY, D. J., SMY, T., AND NAKHLA, M. S. 2005. The creation of compact thermal models of electronic components using model reduction. *IEEE Trans. Adv. Packaging 28,* 2, 240–251.

DHODAPKAR, A., LIM, C. H., CAI, G., AND DAASCH, W. R. 2000. Tem^{2}p^{2}est: A thermal enabled multimodel power/performance estimator. In *Proceedings of the Workshop on Power-Aware Computer Systems*.

DONALD, J. AND MARTONOSI, M. 2006. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the 33rd International Symposium on Computer Architecture*, 78–88.

FLOORPLAN, N. 2007. In *Proceedings of the http://www.chip-architect.com/news/Northwood_130nm_die_text_1600x1200.jpg*.

GUNTHER, S. H., BINNS, F., CARMEAN, D. M., AND HALL, J. C. 2001. Managing the impact of increasing microprocessor power consumption. *Intel Technol. J*.

HE, L., LIAO, W., AND STAN, M. R. 2004. System level leakage reduction considering the interdependence of temperature and leakage. In *Proceedings of the Design Automation Conference*, 12–17.

HEO, S., BARR, K., AND ASANOVIC, K. 2003. Reducing power density through activity migration. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 217–222.

HUANG, W., HUMENAY, E., SKADRON, K., AND STAN, M. R. 2005. The need for a full-chip and package thermal model for thermally optimized IC designs. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 245–250.

HUANG, W., STAN, M. R., SKADRON, K., SANKARANARAYANAN, K., GHOSH, S., AND VELUSAMY, S. 2004. Compact thermal modeling for temperature-aware design. In *Proceedings of the 41st Annual Conference on Design Automation*, 878–883.

INTEL. 2007a. Intel c++ compiler 9.0 for linux. http://www.intel.com/cd/software/products/asmo-na/eng/compilers/clin/index.htm.

INTEL. 2007b. Intel match kernel library 8.0. http://www.intel.com/cd/software/products/asmo-na/eng/perflib/mkl/index.htm%/.

INTEL. 2002. Intel pentium 4 processor in the 478-pin package thermal design guidelines. http://developer.intel.com/design/pentium4/guides/249889.htm.

INTEL. 2004. The microarchitecture of the Intel Pentium 4 processor on 90nm technology. *Intel Technol. J. 8,* 1 (Feb.).

ISCI, C. AND MARTONOSI, M. 2003. Runtime power monitoring in high-end processors: Methodology and empirical data. In *Proceedings of the 36th Annual International Symposium on Microarchitecture*, 93–104.

ISCI, C. AND MARTONOSI, M. 2006. Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques. In *Proceedings of the 12th International Symposium on High-Performance Computer Architecture*, 122–133.

JOSEPH, R. AND MARTONOSI, M. 2001. Run-Time power estimation in high-performance microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design*, 135–140.

KURSUN, E., CHER, C.-Y., BUYUKTOSUNOGLU, A., AND BOSE, P. 2006. Investigating the effects of task scheduling on thermal behavior. In *Proceeding of the 3rd Workshop on Temperature-Aware Computer Systems,* (held in conjunction with ISCA-33).

LEE, K.-J. AND SKADRON, K. 2005. Using performance counters for runtime temperature sensing in high-performance processors. In *Proceedings of the Workshop on High-Performance, Power-Aware Computing (HP-PAC)* (in conjunction with the International Parallel and Distributed Processing Symposium).

LI, H., LIU, P., QI, Z., JIN, L., WU, W., TAN, S. X.-D., AND YANG, J. 2005a. Efficient thermal simulation for run-time temperature tracking and management. In *Proceedings of the Internationl Conference on Computer Design (ICCD)*, 130–133.

LI, Y., BROOKS, D., HU, Z., AND SKADRON, K. 2005b. Performance, energy, and thermal considerations for smt and cmp architectures. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 71–82.

LI, Y., LEE, B., BROOKS, D., HU, Z., AND SKADRON, K. 2006. CMP design space exploration subject to physical constraints. In *Proceedings of the 12th IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

LIU, P., QI, Z., LI, H., JIN, L., WU, W., TAN, S. X.-D., AND YANG, J. 2005. Fast thermal simulation for architecture level dynamic thermal management. In *Proceedings of the IEEE/ACM International Conf. on Computer-Aided Design (ICCAD)*, 639–644.

MONFERRER, P. C., MAGKLIS, G., GONZALEZ, J., AND GONZALEZ, A. 2005. Distributing the frontend for temperature reduction. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 61–70.

PILLAGE, L. T. AND ROHRER, R. A. 1990. Asymptotic waveform evaluation for timing analysis. *IEEE Trans. Comput. Aided Des. 9,* 4.

POWELL, M. D., GOMAA, M., AND VIJAYKUMAR, T. N. 2004. Heat-and-Run: Leveraging SMT and CMP to manage power density through the operating system. In *Proceedings of the 11th International Conference on Architectural Support for Programming Languages and Operating Systems*, 260–270.

RUDNYI, E. B. AND KORVINK, J. G. 2005. Model order reduction for large scale engineering models developed in ANSYS. Lecture Notes in Computer Science, Springer, 349–356.

SENG, J. S. AND TULLSEN, D. M. 2003. The effect of compiler optimizations on Pentium 4 power consumption. In *Proceedings of the 7th Conference on Interaction Between Compilers and Computer Architectures*, 51–56.

SHERWOOD, T., PERELMAN, E., HAMERLY, G., AND CALDER, B. 2002. Automatically characterizing large scale program behavior. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 45–57.

SKADRON, K., ABDELZAHER, T., AND STAN, M. R. 2002. Control-Theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, 17–28.

SKADRON, K., STAN, M. R., HUANG, W., VELUSAMY, S., SANKARANARAYANAN, K., AND TARJAN, D. 2003. Temperature-Aware microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture*, 2–13.

SMSC. 2007. Emc6d100/emc6d101 environmental monitoring and control device with automatic fan capability. http://www.smsc.com/main/tools/discontinued/6d100.pdf.

SRINIVASAN, J. AND ADVE, S. V. 2003. Predictive dynamic thermal management for multimedia applications. In *Proceedings of the 17th Annual International Conference on Supercomputing*, 109–120.

SRINIVASAN, J., ADVE, S. V., BOSE, P., AND RIVERS, J. A. 2006. The case for lifetime reliability-aware microprocessors. In *Proceedings of the 31st International Symposium on Computer Architecture*, 276–287.

STOER, J. AND BULIRSCH, R. 1991. *Introduction to Numerical Analysis*, 2nd ed. Springer.

TSAI, C.-H. AND KANG, S.-M. 2000. Fast temperature calculation for transient electrothermal simulation by mixed frequency/time domain thermal model reduction. In *Proceedings of the 37th Design Automation Conference*, 750–755.

WANG, T.-Y. AND CHEN, C. C.-P. 2004. Spice-Compatible thermal simulation with lumped circuit modeling for thermal reliability analysis based on modeling order reduction. In *Proceedings of the 5th International Symposium on Quality Electronic Design*, 357–362.

WEISSEL, A. AND BELLOSA, F. 2004. Dynamic thermal management for distributed systems. In *Proceedings of the 1st Workshop on Temperature-Aware Computer Systems*.

WU, W., JIN, L., YANG, J., LUI, P., AND TAN, S. 2006. Efficient method for functional unit power estimation in modern microprocessors. In *Proceedings of the 43rd IEEE/ACM Design Automation Conference*.

Received September 2006; revised March 2007; accepted March 2007