USING ON-CHIP EVENT COUNTERS FOR HIGH-RESOLUTION, REAL-TIME TEMPERATURE MEASUREMENT[1]

Sung Woo Chung† and Kevin Skadron‡

Division of Computer Science and Engineering,†
Korea University, Seoul 136-713, Korea
swchung@korea.ac.kr

Department of Computer Science, ‡
University of Virginia, Charlottesville, VA 22901, USA
skadron@cs.virginia.edu

ABSTRACT

This paper proposes a technique to use on-chip event- or performance-counters to augment, or even replace, traditional analog CMOS temperature sensors. Using activity data from the performance counters, energy consumption that consequently causes heat dissipation can be tracked. Simple regression analysis permits us to find a relation between activity data and temperature. Performance counters already exist in many processors for debugging and performance characterization, require only minimal computation to interpret for temperature monitoring, and these calculations only need to operate at low frequency, so the marginal cost of this additional temperature-sensing capability is negligible. Performance counters monitor activity data (access count) of most on-chip functional units and therefore allow high-resolution, localized temperature sensing across a microprocessor. This in turn allows tracking of localized hotspots. Fine-grained, localized sensing is needed because different units can become hotspots depending on benchmarks. This is especially true if a malicious program intentionally induces high activity in a selected functional unit. This paper presents measurements from a commercial system to illustrate the accuracy of performance counters as additional temperature sensors.

KEY WORDS: thermal management, localized hotspot, temperature measurement, performance counter

INTRODUCTION

Technology trends are packing transistors ever more tightly, and as voltage scaling slows down, power densities are increasing rapidly. This in turn creates cooling challenges, especially in high-performance ICs. Possible failure due to high temperature can cause tremendous damage not only to microprocessors but also to whole systems. Effective and robust cooling solutions are a (costly) necessity. Aggressive packaging and fans are the primary solution, but *dynamic thermal management* (DTM) is a complementary solution that allows less expensive packaging/fan ("external") solutions. The external solution can be designed for *typical* operation. Then, if severe environmental conditions or an unusual workload cause temperature to exceed safe operating limits, DTM slows execution to reduce power density until temperatures return to a safe level. DTM has primarily been achieved with voltage/frequency scaling or some mechanism that stops issuing instructions, variously called clock gating, fetch gating, pipeline throttling, etc. In short, these hardware solutions trade reduced performance for reduced packaging cost. These DTM techniques are actually more sophisticated, more aggressive versions of prior runtime techniques that were needed as failsafes in case of catastrophic events like fan failure or heat-sink detachment.

For accurate temperature measurements, on-chip thermal sensors were adopted in modern processors. For example, there are two independent thermal sensors in the Intel Pentium 4 Processor in the 478-pin package [1]. By using an accurate on-die temperature sensing circuit and a fast-acting temperature control circuit, the processor can rapidly initiate thermal management control. The Pentium 4, however, only uses one of its sensors for thermal management; the other is for external use. In fact, hotspots may move over time, depending on which on-chip structures (register file, integer arithmetic, floating-point arithmetic, etc.) are most heavily used. Temperature differences become exponentially larger with distance, so a single sensor is not sufficient to track temperature changes across a large chip like the Pentium 4. This is especially a concern in the case of malicious software that might try to create a hotspot in an un-monitored structure [2][12]. In the case of the Pentium 4, the sensor associated with the thermal-control circuit is placed near the structure that is likely to be the hottest. While this has been verified to be safe for the Pentium 4, it requires a massive validation effort to ensure that no hotspots can arise that would be missed by the single sensor. This validation effort can be avoided by using multiple sensors. An alternative is to simply use a large safety margin, making the DTM trigger temperature well below the actual safe operating limit. This incurs a large overhead in unnecessary throttling that again can be avoided by using multiple sensors.

Using multiple CMOS sensors for fine-grained temperature sensing has its own drawbacks. Sensors must be fairly large to achieve good precision, making them costly in terms of area and power. Increasing the number of sensors also increases the testing and calibration cost for each chip. These concerns have led most chip designs to stick with a small number of sensors.

Software can also benefit from precise, localized temperature sensing capability. For example, some research [3][4][5][6] has explored temperature-aware scheduling to meet real-time or other quality-of-service requirements. Other research, e.g.

[1] This work was performed, while Sung Woo Chung was visiting the LAVA Lab at University of Virginia.

[7][8], has taken advantage of the fact that, even though hotspots move over time, entire micro-architectural structures tend to exhibit approximately the same temperature, because the entire structure operates as a single unit. These techniques assume accurate temperature measurement. In fact, most chips today do not provide this capability.

This paper proposes an accurate, localized temperature-measurement technique which does not incur any complicated computation and does not use additional physical thermal sensors. Instead, it utilizes off-line regression analysis to interpret readings from existing, on-chip performance counters, which can be configured to gather specific micro-architectural events such as cache hits and are therefore a good estimate of processor activity. Performance counters have been proposed before as a way to infer temperature [9][10][6], but these prior techniques required expensive, on-line computation, while our regression-based approach incurs almost no overhead. At runtime, the processor only has to calculate the temperature from a simple formula by reading performance counter values and performing a few multiplies and adds. Compared to fine-grained *hardware* temperature sensors, performance counters trade a negligible extra computational requirement and some imprecision in timing for a substantial reduction in hardware cost.

The rest of this paper is organized as follows. Section 2 explains previous work on temperature measurement. Section 3 describes our regression-based temperature measurement technique. Section 4 evaluates the accuracy of the proposed technique on two different architectural functional units. Section 5 concludes the paper and describes some avenues for future work.

PEVIOUS WORK ON SOFTWARE TECHNIQUES FOR TEMPERATURE MEASUREMENT

Initially, the performance counter was designed to evaluate the performance of each functional unit. In recent microprocessors, including even embedded processors, most of them embed performance counters. In this paper, the performance counters are utilized to obtain activity data (access counts) of functional units. For example, activity data for the cache can be obtained by reading the performance counter, since the activity data is stored in the performance counter in order to calculate cache miss rate (= cache miss/cache access). In Pentium 4, activity data for most functional units can be read through as many as 45 performance counters.

HotSpot is a software model for simulating on-chip temperatures at a micro-architectural granularity. It models the processor as a network of thermal resistors and conductors per functional unit, where power dissipation in each unit is treated as current source in the RC network [7][8]. This model gets activity data not from performance counters but from simulations. Lee and Skadron extended HotSpot to interface with performance counters in order to get activity data from a real processor [10]. The HotSpot calculations are, however, too expensive to be practical for real-time temperature estimation.

This creates an online thermal model that can measure temperature based on real workloads in real processors. Their approach estimates power dissipation from performance counters based on a model by Isci and Martonosi [11]. To better understand the need for localized temperature measurement, Lee and Skadron also proposed an analytical model describing the maximum temperature difference between a hotspot and a region of interest based on their distance and attributes of the processor package [12]. Bellosa et al. also use performance counters for an event-driven approach [9][3]. They calculate a single temperature value by solving differential equations, which occupies substantial processor resources.

In this paper, we propose a simple and accurate temperature-estimation technique based on performance counters that require minimal computation.

SIMPLE AND ACCURATE TEMPERATURE MEASUREMENT TECHNIQUE

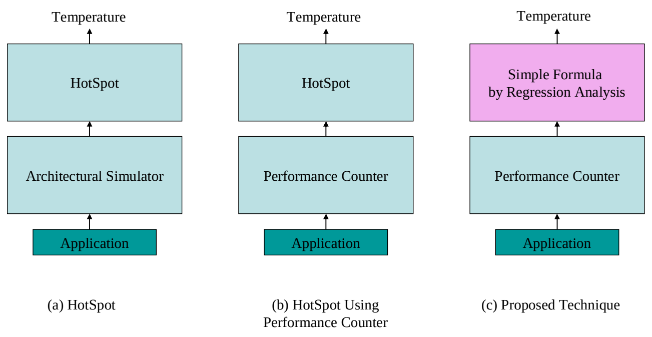


Fig. 1 Comparison of the original HotSpot, the performance-counter version of HotSpot, and our proposed technique

As mentioned earlier, previous techniques for estimating temperature via software techniques, shown in Figure 1 (a) and Figure 1 (b), are accurate but impose too much computational overhead to be practical at runtime, since they use the HotSpot or differential equations to calculate temperature. In this paper, we propose a simple temperature measurement technique, shown in Figure 1 (c). In our proposed technique, we also read the performance counters for activity data of each functional unit of interest. But in contrast to prior techniques, we use simple *offline* regression analysis to find a simple relation between selected values of activity data and observed values of temperature. Please recall that the most probable value of Y can be predicted for any value of X by simple regression analysis. Temperature can then be estimated using a simple formula ($T = aX + b$, where T is temperature, X is activity data, and a and b are coefficients). In most cases, we only have to consider only the activity count of the functional unit that is investigated. The key observation is that the regression captures second-order contributions from other functional units.

At runtime, multiplying the activity count by the regression coefficient is required for temperature measurement. Although it is feasible to re-compute temperature every cycle, this is wasteful, since even at the fine granularity of architectural units, temperatures take at least 100K cycles to rise by 0.1C [7]. We chose a sample period 10 ms, since this is the scheduling granularity of commercial operating systems and creates a natural opportunity for software to read the event counters. For our CPU clock rate of 2.6 GHz, this works out to be sampling period of 26 M cycles. This is in any case the minimum granularity at which software techniques could perform any kind of thermal management.

For example, to compute the temperature of the integer register file, we only use the IIPC performance-counter statistic (Integer Instructions Per Cycle). We did try multiple regression analysis with the current IIPC and the previous IIPC. Results were at best minimally improved compared to results from simple regression analysis, and in fact accuracy with multiple regression was sometimes worse.

EVALUATION

Experimental Environment

The processor used for the experiments is a 2.6 GHz Pentium 4, 130 nm Northwood core. The typical power dissipation is 69.0 W, and the operating voltage is 1.6 V [12]. The processor supports hyper-threading technology, which allows the processor to run two threads simultaneously. This means that the task that regularly reads the performance counters and calculates the temperature interferes minimally with user tasks: not only does it consist of only a few instructions, but hyper-threading fits these few instructions into empty execution slots as instructions are issued within the processor.

The Pentium 4 includes a rich set of performance monitoring features, with 45 configurable events and 18 physical performance counters [13][14]. The performance counters are used to count specific micro-architectural events for debugging and performance measurements. Each counter is associated with one counter configurable control register (CCCR), which determines the specific counting scheme. The event selection control registers (ESCRs) determine which event is to be counted. A simplified device driver, adapted from the abyss device driver[14], is used to configure all the control registers and read the performance counters.

The temperature model requires the geometric specifications and the floorplan layout of the processor. Table 1 shows the parameters that we derived to configure HotSpot for the Pentium 4. These parameters are based on design schematics found in [12]. We also use the floorplan layout that was adapted from the Northwood core die photo [15].

Table 1. HotSpot simulator configuration to approximate a Pentium 4

HotSpot Variable	Value	Description (Unit)
t_chip	0.5	Chip thickness (mm)
c_convec	140.4	Convection capacitance (J/K)
r_convec	0.1	Convection resistance (K/W)
s_sink	60	Heat sink thickness (mm)
t_sink	6.9	Heat sink thickness (mm)
s_spreader	30	Heat spreader side (mm)
t_spreader	1	Heat spreader thickness (mm)
t_interface	0.075	Interface material thickness (mm)
ambient	40+273.15	Ambient temp (K) (inside box)

We selected ten benchmarks (bzip2, crafty, gap, gcc, gzip, mcf, parser, twolf, vortex, and vpr) from the SPEC CPU2000 benchmark suite [16]. The SPEC CPU2000 benchmark suite is designed to provide performance measurements that can be used to compare computation-intensive workloads on different computer systems. More details on ten benchmarks are described in Table 2. By running all the benchmarks, we obtain the regression coefficients. For regression analysis, we ran 50G cycles (5K samples) for each benchmark. We did a regression analysis with the collected data and we compared the proposed technique with the performance-counter version of HotSpot [10] that was previously validated against the original, validated HotSpot [8].

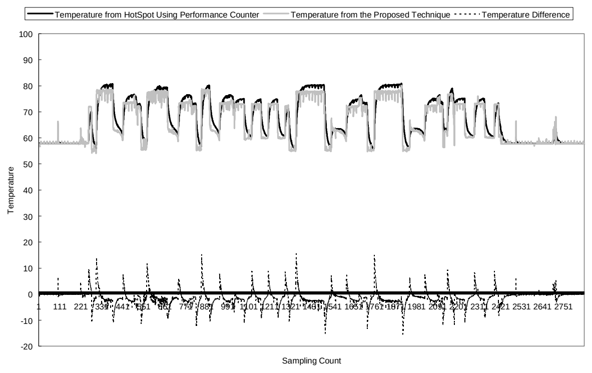
Table 2. SPEC CPU2000 Benchmarks used for evaluation

Benchmark	Description
Bzip2	Compression program based on Julian Seward's bzip2 version 0.1
Crafty	High-performance Computer Chess program that is designed around a 64-bit word
Gap	Program for computational discrete algebra, with particular emphasis on Computational Group Theory
Gcc	Gcc compiler with many of its optimization flags enabled
Gzip	Compression program that uses Lempel-Ziv coding (LZ77) as its compression algorithm
Mcf	Program used for single-depot vehicle scheduling in public mass transportation
Parser	Syntactic parser of English, based on link grammar, an original theory of English syntax
Twolf	Placement and global routing package used in the process of creating the lithography artwork needed for the production of microchips
Vortex	Single-user object-oriented database transaction benchmark
Vpr	Integrated Circuit Computer-Aided Design Program that performs placement and routing in Field-Programmable Gate Arrays

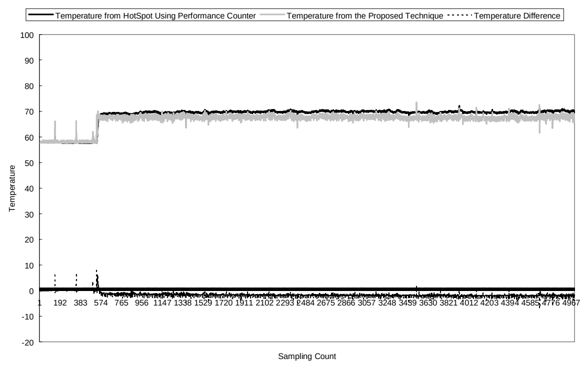
Because the Pentium 4's externally-visible temperature sensor is not helpful (being located on the edge of the chip, far from structures of interest), we use the performance-counter version of HotSpot [10] as the source of reference temperatures for determining accuracy, and for calculating the regression coefficients.

Temperature Measurement on the Integer Register File

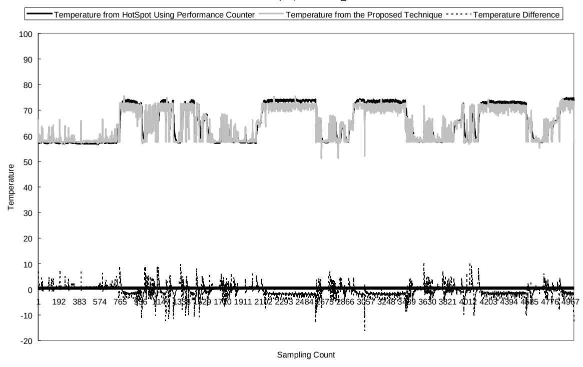
We investigate the temperature of the integer register file which is known as one of the hottest functional units. In the simple regression analysis formula, IIPC is X (selected value) and the temperature is Y (observed value). The integer register file is accessed whenever an instruction is issued unless the instruction includes a floating point operation. Thus, we use IIPC to measure the integer register file. The formula we obtained from the simple regression analysis is $Y = 14.92*X + 50.39$. Figure 2 shows the temperature of the integer register file from the performance-counter version of HotSpot, from the proposed technique, and the temperature difference. When the temperature difference is a positive number, the temperature from the proposed technique is overestimated, and vice-versa.



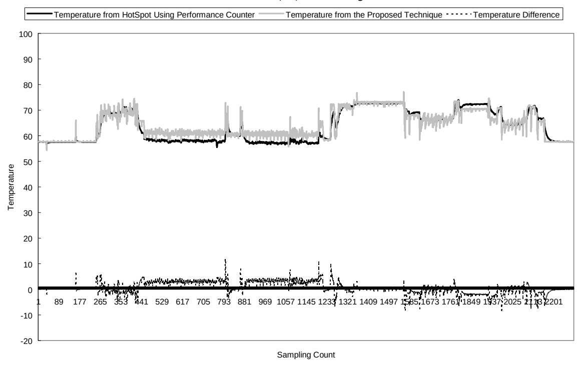
(a) bzip2



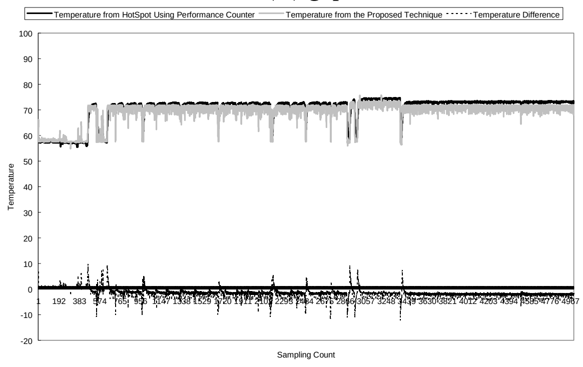
(b) crafty



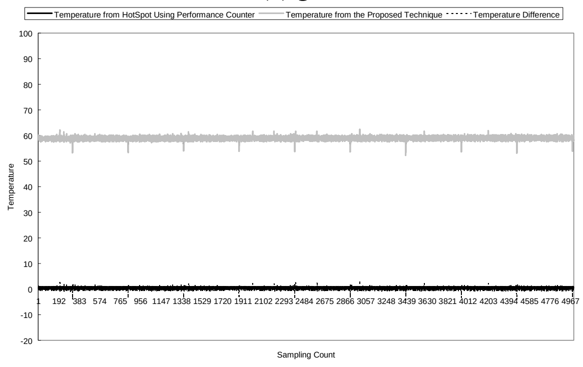
(c) gap



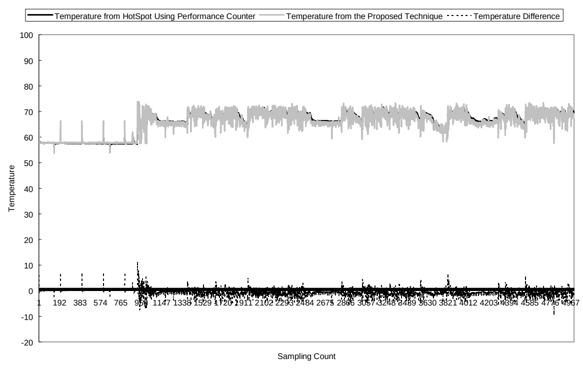
(d) gcc



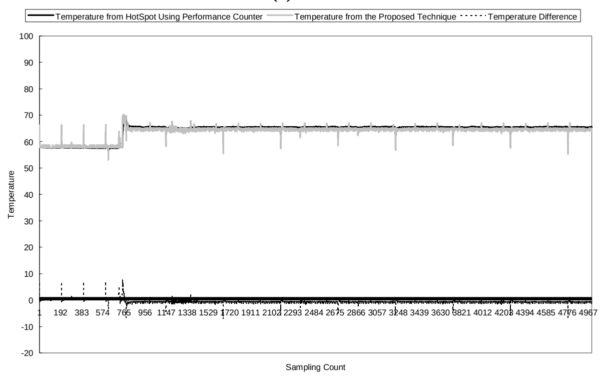
(e) gzip



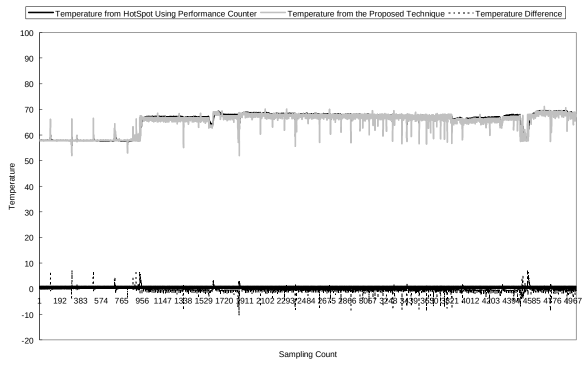
(f) mcf



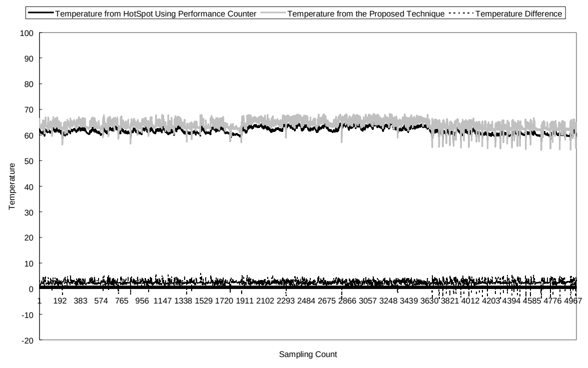
(g) parser



(h) twolf



(i) vortex



(j) vpr

Fig. 2 Accuracy of integer register file temperature measurement (temperature difference = temperature from the proposed technique – temperature from performance-counter-HotSpot)

Although there are times when our method under- or over-estimates temperature by as much as 10°, these large differences only occur when the performance-counter technique responds faster than the actual temperature. The performance-counter technique under-estimates *peak* temperature by at most 2.4°!

Bzip2 provides an excellent illustration, since the spikes and dips are most severe in bzip2. Figure 3 magnifies a portion of its execution. As shown at points (i) and (ii) in Figure 3, when the temperature increases/decreases fast, we can observe the temperature from the proposed technique increases/decreases much faster. The reason is that our technique considers only activity data (in this case, the IIPC for the integer register file), and our temperature is linearly proportional to the activity data. However, the temperature from the Hotspot is changed slowly after the activity data (in this case, IIPC) is changed, because it takes some time for the dynamic power to be transformed into thermal dissipation. As far as we can tell, all the major spikes and dips are due to this reason. This timing inaccuracy can actually be useful, because it acts like a temperature predictor. In fact, huge body of architecture research studies how predictors of various types (e.g. branch, value predictors) allow architectural optimizations.

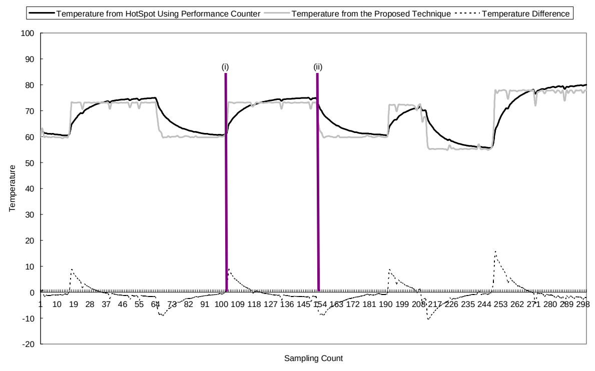
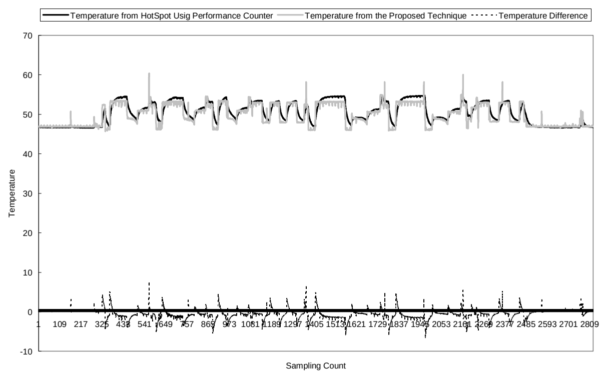


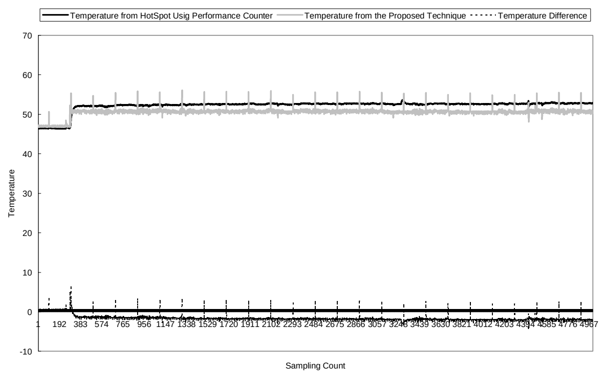
Fig. 3 Accuracy of temperature measurement in bzip2 (enlarged figure)

Temperature Measurement on the L1 Data Cache

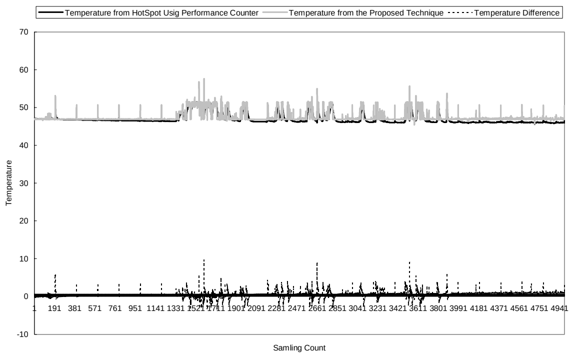
For the L1 data cache, we use the number of L1 data cache accesses as X for the simple regression analysis. The number of L1 data cache accesses is the summation of read data allocation from lower-level memory, write data allocation from lower-level memory, and total requests from the pipeline to the L1 data cache. Read /write data allocations occur when there is a cache miss in the L1 data cache. For the simple



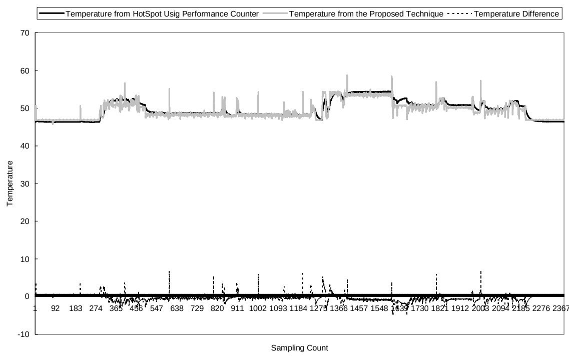
(a) bzip2



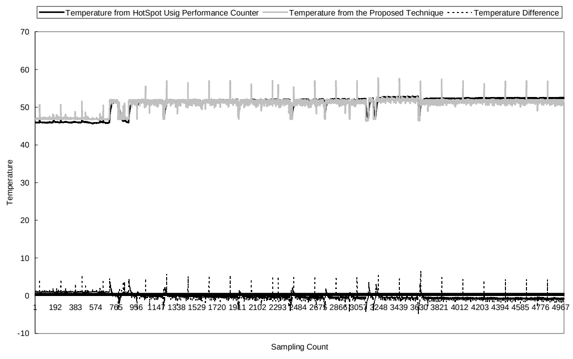
(b) crafty



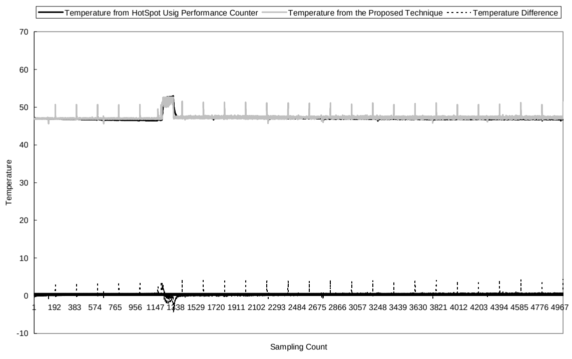
(c) gap



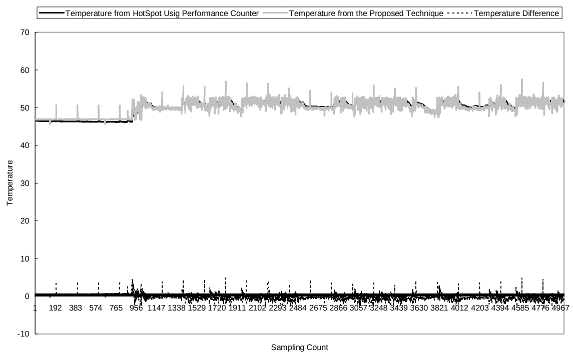
(d) gcc



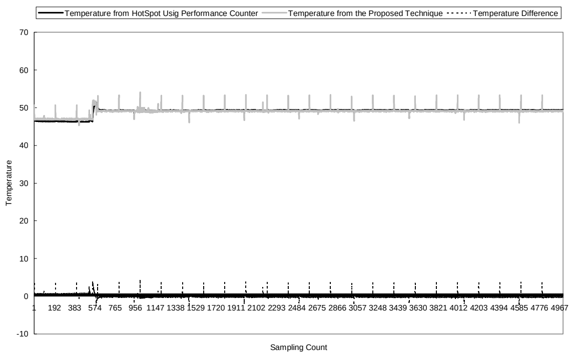
(e) gzip



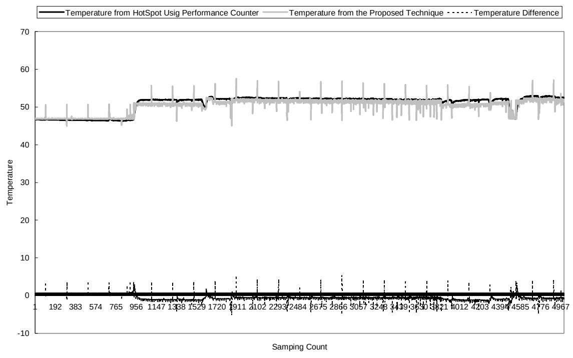
(f) mcf



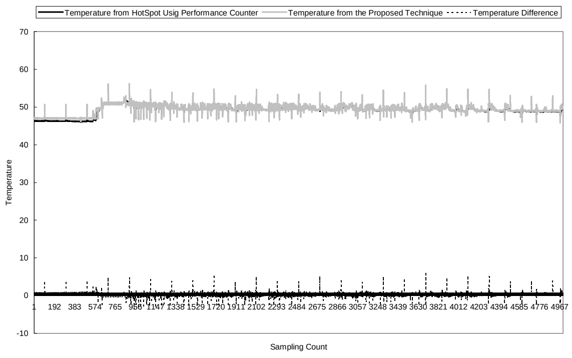
(g) parser



(h) twolf



(i) vortex



(j) vpr

Fig. 4 Accuracy of L1 data cache temperature measurement (temperature difference = temperature from the proposed technique – temperature from performance-counter-HotSpot)

regression analysis, X is the number of L1 data cache accesses and Y is the temperature. The formula we obtained from the simple regression analysis is $Y = 1.4*10^{-6}*X + 44.413$. The coefficient for X is much smaller here than with the register file, because the register-file temperature uses IIPC (which is typically between 0 and 4) while the data cache temperature uses number of data references, which is in the millions.

Figure 4 shows the temperature of the L1 data cache from HotSpot, the temperature from the proposed technique, and the temperature difference. In most cases, the temperature difference is small. In addition to the spikes and dips shown in Figure 2, there are periodic spikes. The reason is periodic large number of L1 data cache accesses, which may be caused by operating system context-switching activities. The spikes of the estimated temperature from the performance counter can be somewhat eliminated by limiting the maximum increase/decrease of the estimated temperature from the performance counter in one sampling period.

CONCLUSIONS AND FUTURE WORK

Temperature is typically similar across an architectural functional unit, such as a register file or computational unit, because the whole unit operates in unison. Uneven activity from one functional unit to another, however, results in localized hotspots, and the hotspot may move over time. Accurate thermal monitoring therefore requires thermal sensors near any potential hotspot. This may be too costly, because precise CMOS temperature sensors are expensive in terms of area and power. As an alternative, we propose a high-resolution, real-time temperature measurement technique using on-chip performance (event) counters. The proposed technique uses simple regression analysis using X (activity data) to calculate Y (temperature). The advantage of these "soft sensors" based on performance counters is that the counters are already embedded in most processors.

Experimental results, obtained from the Intel Pentium 4 processor, show that using performance counters for temperature measurement estimates peak temperature with high accuracy. Counter-based sensing can therefore augment a small number of hardware sensors to provide accurate, localized temperature sensing with low hardware and execution-time costs. Performance-counter based techniques are especially well suited for software thermal-management techniques like temperature-aware job scheduling.

Interesting directions for future work include techniques for improving both the temperature and timewise accuracy of the soft-sensor approach, as well as applications in temperature-aware scheduling. Temperature-aware scheduling is becoming increasingly important with the advent of chips with multiple CPUs, for which the scheduling problem requires not only how to order tasks in time, but how to place them and possibly migrate them across the multiple CPUs.

Acknowledgements

This work was funded in part by the National Science Foundation under grant nos. CAREER CCR-0133634, CCF-0429765, the Army Research Office under grant no. W911NF-04-1-0288, a Faculty Partnership Award from IBM T.J. Watson, a gift from Intel MTL, an Excellence Award from the Univ. of Virginia Fund for Excellence in Science and Technology, and the IT National Scholarship Program from IITA & MIC, Korea.

References

[1] J. Citaerlla. The Intel PIV's Thermal Diodes. Available in http://www.overclockers.com/artocles 517.

[2] K.-J. Lee and K. Skadron. Analytical Model for Sensor Placement on Microprocessors. In Proceedings of the IEEE International Conference on Computer Design (ICCD'05), Oct. 2005.

[3] A. Merkel, F. Bellosa, and A. Weissel. Event-Driven Thermal Management in SMP Systems, In Proceedings of the Second Workshop on Temperature-Aware Computer Systems (TACS'05), June 2005.

[4] M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run : Leveraging SMT and CMP to Manage Power Density Through the Operating System. In Proceedings of International Conference on Architectural Support for Programming Language and Operating System (ASPLOS'04), Oct. 2004.

[5] J. Srinivasan and S. V. Adve. Predictive Dynamic Thermal Management for Multimedia Applications. In Proceedings of International Conference on Supercomputing (ICS'03), June 2003.

[6] A. Weissel and F. Bellosa, Dynamic Thermal Management for Distributed Systems. In Proceedings of the First Workshop on Temperature-Aware Computer Systems (TACS'04), June 2004

[7] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayana, and D. Tarjan. Temperature-Aware Microarchitecture. In Proceedings of the 30th International Symposium on Computer Architecture (ISCA'03), June 2003.

[8] K. Skadron, M. Stan, K. Sankaranarayana, W. Huang, S. Velusamy, and D. Tarjan. Temperature-Aware Microarchitecture: Modeling and Implementation. ACM Transaction on Architecture and Code Optimization. Vol. 1, No. 1, March 2004, pp. 94-125.

[9] F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-Driven Energy Accounting for Dynamic Thermal Management. In Proceedings of COLP 2003, Sep. 2003.

[10] K.-J. Lee and K. Skadron. Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors. In Proceedings of the Workshop on High-Performance, Power-Aware Computing (HP-PAC), April 2005.

[11] C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical data. In Proceedings of Microarchitecture (Micro'03), Dec. 2003.

[12] Intel Pentium 4 Technical Documents. Available in http://www.intel.com/design/Petium4/ documentation.html

[13] B. Sprunt. Pentium 4 Performance-Monitoring Features. IEEE Micro, 22(4), Jul/Aug 2002.

[14] B. Sprunt. Brink and Abyss Pentium 4 Performance Counter Tools for Linux. Available in http://www.eg.bucknell.edu/bsprunt/emon/brink_abyss.

[15] Intel Pentium 4 Northwood Die Photo. Available in http://www.chip-architect.com/news/ 2003_04_20_Looking_at _Intels_Prescott_part2.html

[16] Standard Performance Evaluation Corp.. Available in http://www.specbench.org.