HybDTM: A Coordinated Hardware-Software Approach for Dynamic Thermal Management [*]

Amit Kumar[†], Li Shang[‡], Li-Shiuan Peh[†] and Niraj K. Jha[†]
[†]Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544
[‡]Dept. of Electrical and Computer Engineering, Queen's University, Kingston ON K7L3N6
[†]{amitk, peh, jha}@princeton.edu, [‡]li.shang@queensu.ca

ABSTRACT

With ever-increasing power density and cooling costs in modern high-performance systems, dynamic thermal management (DTM) has emerged as an effective technique for guaranteeing thermal safety at run-time. While past works on DTM have focused on different techniques in isolation, they fail to consider a synergistic mechanism using both hardware and software support and hence lead to a significant execution time overhead.

In this paper, we propose HybDTM, a methodology for fine-grained, coordinated thermal management using a hybrid of hardware techniques, such as clock gating, and software techniques, such as thermal-aware process scheduling, synergistically leveraging the advantages of both approaches. We show that while hardware techniques can be used reactively to manage thermal emergencies, proactive use of low-overhead software techniques can rely on application-specific thermal profiles to lower system temperature. Our technique involves a novel regression-based thermal model which provides fast and accurate temperature estimates for run-time thermal characterization of applications running on the system, using hardware performance counters, while considering system-level thermal issues. We evaluate HybDTM on an actual desktop system running a number of SPEC2000 benchmarks, in both uniprocessor and simultaneous multi-threading (SMT) environments, and show that it is able to successfully manage the overall temperature with an average execution time overhead of only 9.9% (16.3% maximum) compared to the case without any DTM, as opposed to 20.4% (29.5% maximum) overhead for purely hardware-based DTM.

Categories and Subject Descriptors: B.7.2 [**Hardware**]: Design Aids
General Terms: Design, performance
Keywords: dynamic thermal management, thermal model, hybrid hardware-software management

1. INTRODUCTION

Escalating chip complexity and associated power budgets have led to increasing chip temperatures. Higher temperatures adversely affect circuit reliability and lead to timing uncertainties, prompting wider timing margins, and hence degrade performance. Traditional design of cooling packages in high-performance systems for worst-case temperature spikes is no longer cost-effective. This has resulted in the move towards solutions [3] specified for average temperature contribution of the chip, and using DTM techniques to tackle infrequent occurrences in which the chip temperature exceeds the thermal limit supported by the packaging and cooling solution.

Researchers in the recent past have proposed a number of DTM techniques using either hardware or software mechanisms [5, 6, 16]. While hardware-based techniques, such as dynamic voltage scaling (DVS), clock gating, fetch toggling, etc., are effective in managing temperature, they incur a high execution time overhead. Moreover, they ignore application-specific information. Hence, in case of thermal emergencies, all applications are penalized equally and suffer an equal impact in performance. Software-based techniques using an operating system (OS), such as energy-aware process scheduling, on the other hand, have a lower performance impact, but cannot guarantee thermal safety. This calls for a *hybrid* approach towards managing overall temperature, which uses both hardware-based and software-based DTM techniques in a coordinated fashion to tackle thermal emergencies with much lower execution time overhead.

Implementing a hybrid DTM technique using OS support requires a thermal model which can provide fast and accurate temperature predictions without imposing a large overhead on the critical path. Prior work on architecture-level thermal modeling [5,14,16] has mostly focused on using power models first to estimate the power consumption of different microarchitectural units and then feeding it as input to a complex thermal model for estimating temperature values, requiring multiple iterations. Hence, such approaches are inherently slow and incur a large overhead each time they are invoked.

In this paper, we first develop a novel regression-based thermal model for providing fast and accurate predictions of the processor temperature *directly* from hardware performance counters found in most modern processors, e.g., Pentium 4 [1]. Having extremely low overhead, our model can be integrated inside the OS for doing fine-grained DTM. Moreover, it is able to account for system-wide effects by considering run-time variations in the ambient temperature due to the thermal contribution of other system components, such as the memory module. Using this model, we keep track of both the overall processor temperature as well as the thermal contribution of each application running on the system. We then propose HybDTM, a hybrid hardware-software DTM technique using both proactive software mechanisms, such as thermal-aware scheduling, to avert thermal emergencies and reactive hardware mechanisms, such as clock gating, to deal with thermal emergencies when they occur. We implemented our proposed technique on an actual desktop machine with an Intel Pentium 4 processor running under a modified Linux 2.6.9 OS in both uniprocessor and SMT configurations and evaluated its effectiveness using a range of SPEC2000 benchmarks. We present a comparative study analyzing the pros and cons of purely software-based DTM, purely hardware-based DTM and HybDTM, and show that HybDTM is able to effectively manage the overall temperature with an average execution time overhead of only 9.9% (16.3% maximum) compared to the case without any DTM, while the corresponding average overhead for hardware-based DTM is 20.4% (29.5% maximum). A brief summary of our contributions is as follows:

- A novel low-overhead hybrid DTM technique using

[*]This work was supported by NSF under Grant No. CCR-0324891.

hardware and software mechanisms in a synergistic fashion.
- A new thermal model providing fast and accurate predictions of the overall processor temperature directly from hardware performance counters, while considering system-wide temperature variations.
- A modified thermal-aware OS to do online thermal characterization of each process as well as the entire system as a whole in both uniprocessor and SMT environments.
- Analysis of thermal issues at the system level considering other system components, such as the memory, along with the processor.

2. OVERVIEW

Fig. 1 presents the high-level overview of our proposed hybrid DTM implementation.

Hardware setup: The desktop machine used in our study is a 3.2GHz Pentium 4 processor, with a 8KB L1 and 512KB L2 cache, and 800MHz front side bus, supporting hyper-threading technology. Pentium 4 supports an on-die thermal diode [2] whose value can be read using an external thermal sensor. We use this feature to measure the processor temperature at periodic intervals. The processor has 18 hardware counters which are capable of counting up to 18 hardware events simultaneously. We configured these counters to track the usage of different processor microarchitectural units along with the memory module. Pentium 4 also supports software-controlled clock modulation [1], allowing the OS to throttle the processor clock in certain discrete steps to manage processor power and hence temperature. This feature is used by our DTM policy for hardware-directed DTM. The memory used in our system consists of two 512MB Samsung DDR400 SDRAM modules, used in a dual-channel configuration. We use an ASUS P4P800SE motherboard which supports a Winbond W83627THF Super I/O chip which is capable of sensing the processor temperature, using the on-die thermal sensor mentioned above, along with the ambient temperature. Temperature values are read from this Super I/O chip through the system management bus (SMBus).

Software setup: The OS used in our setup is a modified Linux 2.6.9 kernel. To make the OS thermal-aware, we integrated our regression-based thermal model into the kernel. Our model uses hardware performance counters as input to gather the thermal profiles of the individual applications running on the system as well as estimate the overall temperature at run-time. Online temperature readings from the hardware sensor are used for periodic training of our model to account for changes in the cooling package. We also implemented our fine-grained hybrid DTM policy inside the kernel for providing software directives to the scheduler for scheduling processes in a thermal-aware fashion as well as hardware directives to the processor for setting the appropriate clock throttling ratio in case of thermal emergencies.

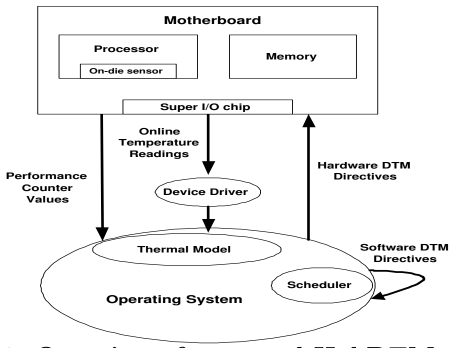

Figure 1: Overview of proposed HybDTM solution.

3. FROM PERFORMANCE COUNTERS TO TEMPERATURE

High power density in modern processors has led to complex cooling designs. Thermal characterization and management requires models which can accurately predict the overall chip temperature taking into consideration these complex cooling package designs as well as the ambient environment.

3.1 Related work in thermal modeling

Prior work on architecture-level thermal modeling for microprocessors [14, 16] has made use of the duality between thermal and electrical circuits by considering the thermal resistances and capacitances of different microarchitectural units and different layers of the cooling package. While these models can be used for early design stage analysis and DTM, their major drawback is that they involve complex calculations and hence incur a large overhead every time they are invoked [13]. Researchers have also looked at using performance counters to model processor power and developing an RC network based thermal model on top of it to estimate temperature [10,18]. This two-step process again has a high overhead and is hence not suitable for obtaining frequent estimates of the processor temperature. Moreover, it fails to account for system-wide thermal effects where the temperature of the processor is affected not just by its own power consumption but by the temperature of other system components, such as the memory. Such effects are prominent in closed systems, such as desktops and laptops, and ignoring them can lead to significant errors in temperature estimation. Using hardware temperature sensors, supported by modern microprocessors, such as Pentium 4 [2], to monitor the run-time temperature, takes a long time (of the order of ms) to obtain each temperature reading. This makes this approach unable to capture transient temperature spikes in the processor temperature, which can change at a much faster rate.

Our approach to characterizing the thermal behavior of different applications and the entire processor, in general, is based on using hardware performance counters, along with a regression-based thermal model. We use performance counters to predict temperature directly, without the use of any complex RC network, which lowers the overhead of our model significantly, thereby making it feasible to integrate it inside the OS for implementing fine-grained DTM.

3.2 Performance counters for thermal characterization of applications

Most modern processors provide performance counters to allow monitoring of specific hardware events for the purpose of debugging and system tuning. The Pentium 4 processor supports 18 hardware performance counters [1], which can be configured using particular model-specific registers (MSRs) to count upto 18 hardware events simultaneously. The details of the processor microarchitecture can be found in [8].

Performance counter setup: The performance monitoring hardware in Pentium 4 broadly consists of event detectors and event counters. The event detectors can be configured to detect several hardware events, such as cache misses, number of instructions retired, number of instruction translation look-aside buffer (ITLB) references, number of data translation look-aside buffer (DTLB) references, etc. A detailed description of the performance monitoring features of Pentium 4 can be found in [1,17].

For characterizing the thermal behavior of each individual application, we first characterize the usage of different microarchitectural units by the application, using a set of 21 hardware events covering different on-chip physical subunits. The different units targeted include the floating point unit, memory order buffer, L1 cache, DTLB, instruction fetch logic, branch predictor unit, floating point register file, ITLB, L2 cache, bus control, retirement logic, microcode ROM, instruction decoder, allocation logic, trace cache, rename logic, instruction queue and instruction scheduler. No direct counter is available for counting the number of integer instructions. However, we can indirectly estimate this number from the total number of instructions and total number of non-integer instructions. Since we use a linear regression model for estimating temperature using performance counters, this linear combination is automatically factored into our model.

3.3 Accounting for system-wide effects

Microprocessor heat dissipation is significantly affected by the ambient temperature, which in a closed system, such as a desktop or laptop, is determined by the power consumption of several system components together. Apart from the processor, the other major component which affects the ambient temperature is the memory module. Modern DDR SDRAM modules can consume large amounts of power which has led to prior work being done on dynamic power management for memories [7]. We developed a simple power and thermal model for memory using the approach followed in [4, 11] to estimate its temperature assuming different usage patterns. We found that although the memory temperature did not reach very high values, with the peak temperature found to be below 52°C, it showed significant variations of upto 8°C depending upon usage. Therefore, it can be seen that the memory temperature can significantly affect the ambient conditions and hence memory usage needs to be accounted for while modeling the processor temperature.

Based on the above observation, we configured one of the counters in our setup, to count the number of read and write transactions on the bus. We used this as a proxy for memory usage and added it as an input to our model, bringing the total number of hardware events monitored to 22.

3.4 Regression analysis

Different microarchitectural units in a processor may contribute differently towards the overall chip temperature. For example, similar usage of the floating-point unit and ITLB by an application may have vastly different impact on the overall temperature. Hence, when using performance counters to estimate temperature, the usage of different units has to be weighted appropriately. We use a regression based approach to find weights corresponding to each of the 22 hardware events used in our model. The basic equation for estimating temperature is as follows.

$$T_{overall} = w_{const} + \sum_{i=1}^{22} w_i \frac{u_i}{t_{total}} \tag{1}$$

where $T_{overall}$ is the overall chip temperature, w_{const} is the temperature contribution due to the energy dissipated by the processor when it is idle, w_i is the regression coefficient corresponding to hardware event i, u_i is the aggregate value of the performance counter for event i, and t_{total} is the total number of processor clock cycles elapsed. Here, w_i is a measure of the thermal contribution of hardware event i towards the overall temperature.

In order to find the regression coefficients mentioned above, we need to consider the entire *regression space*, taking into account the contribution of each hardware event in isolation (to capture the thermal impact of different microarchitectural units) as well as their different possible combinations to capture the interaction between different events (lateral thermal correlation between different units). For this, we wrote a set of 30 thermal microbenchmarks targeting different microarchitectural units or their combinations. To cover the entire regression space, we ran the microbenchmarks individually as well as several combinations of them. During each run, we collected the values of different hardware events using our performance counter setup. This formed the predictor set in our regression equation. We measured the steady-state temperature of each microbenchmark run, using the on-die thermal sensor. For this, we used our hardware setup, as explained in Section 2. The temperature values for each run formed an observation set in our regression equation.

Processor thermal model: We used partial least-squares (PLS) regression to find the coefficients corresponding to different hardware events. PLS minimizes the sum-of-squares of the deviations of different data points from the actual model while explaining as much as possible the covariance between the predictor variables and observations. The regression equation is given as:

$$\begin{pmatrix} 1 & u_{11} & u_{12} & \cdots & u_{1m} \\ 1 & u_{21} & u_{22} & \cdots & u_{2m} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & u_{p1} & u_{p2} & \cdots & u_{pm} \end{pmatrix} \begin{pmatrix} w_{const} \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{pmatrix} = \begin{pmatrix} T_1 \\ T_2 \\ \vdots \\ T_p \end{pmatrix}$$

where u_{ij} denotes the normalized usage of hardware event j by microbenchmark i, w_{const} denotes the constant term, w_j denotes the regression coefficient for hardware event j and T_i denotes the temperature observed for microbenchmark i.

Run-time training: Changes in the cooling and ambient conditions can significantly affect the processor temperature. To adapt our model to such changes, we readjust our regression coefficients by doing selective runs of our microbenchmarks at periodic time intervals, which being infrequent, can be done when the system is idle.

Overhead: We integrated our model inside the Linux kernel and measured its execution time overhead by running a number of microbenchmarks. The execution time was found to increase by less than 2% on average for all the runs.

3.5 Validation

Using the set of microbenchmarks we developed, our analysis showed that the regression coefficients obtained using PLS were able to explain 100% covariance with the predictor set and 93% covariance with the observation set. Fig. 2 shows the measured and estimated steady-state temperature values for a number of microbenchmarks. It can be seen that the steady-state temperature prediction using our regression-based thermal model closely tracks the temperature measured from the on-die thermal diode, with the average error for all runs found to be less than 5%.

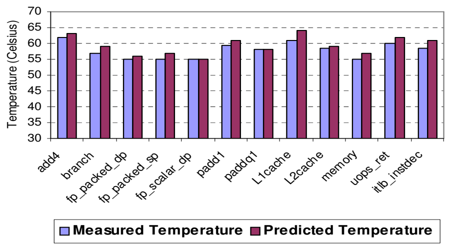

Figure 2: Validation of thermal model for each microbenchmark

3.6 Limitations

Our regression-based thermal model is based on temperature values measured from the on-die thermal diode in Pentium 4. We assume that this value corresponds to the temperature of the hottest unit of the chip or the "hotspot". This limitation is purely processor-specific and in the future as more on-die sensors are added to the processor, the "hotspot" temperature can be measured more accurately. Another limitation of our model is that due to the limited number of counters available for counting floating-point events simultaneously, we use time-multiplexing of counters to estimate the usage of floating-point units. However, since in our setup, we change the counter configuration on every context switch, this error is minimal.

4. HYBRID DTM

In this section, we propose a hybrid DTM policy which makes use of both hardware and software mechanisms in a synergistic fashion to alleviate thermal emergencies with minimal performance impact. The major components of our proposed policy are shown in Fig. 3.

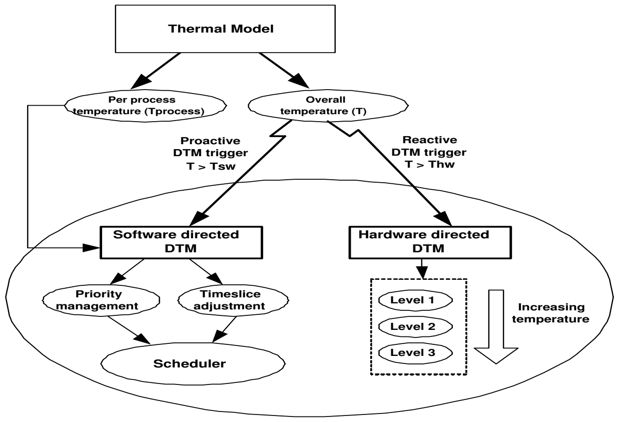

Figure 3: Hybrid DTM flow

4.1 Related work in DTM

DTM techniques [6, 9, 16] used to manage chip temperature employ some mechanism to lower the total energy consumption in case of thermal emergencies. While doing so lowers the overall temperature, this comes at the cost of increased execution time. Furthermore, using different hardware-based DTM techniques, such as DVS, clock gating, fetch toggling, etc., or their combinations [15], affects the performance of all applications equally in case of thermal emergencies, thereby leading to worse overall performance. Different applications running on a system may contribute differently to the overall temperature, depending on the way they utilize the processor resources. For example, consider two applications running on a system – one of them compute-intensive and the other an interactive application. Interactive applications are often not CPU-intensive and remain blocked during a large chunk of their execution time waiting for user response. If the processor temperature crosses the thermal emergency level, a DTM technique, which is not application-aware, will penalize both these applications equally, thereby leading to higher user-perceived latency for the interactive application. While software-based DTM can factor in application-specific thermal behavior [18], a coordinated and fine-grained use of both hardware and software support is needed to minimize the performance impact while guaranteeing thermal safety.

4.2 Run-time thermal characterization

Our approach towards characterizing the thermal contribution of individual processes involves gathering the usage pattern of different processor microarchitectural units at run-time and then using this as an input to our thermal model to directly estimate temperature. For each process, we keep track of the short-term (local) and long-term (global) history of temperature values. The local temperature t_{lp} captures the transient variations in process temperature and is equal to the temperature contribution of the process when the last time it ran on the processor before being context switched out by a different process. On the other hand, the global temperature t_{gp} is calculated based on the aggregate of the counter values for each process and measured from the time the process started. The overall temperature contribution of the process is then given as:

$$T_{process} = w_{lp}t_{lp} + w_{gp}t_{gp} \tag{2}$$

where $T_{process}$ is the overall temperature contribution of the process, and w_{lp} and w_{gp} the corresponding weights given to the local and global temperatures. w_{lp} is given a higher value than w_{gp} to capture transient variations in the process temperature.

A similar method is followed to estimate the overall temperature of the entire chip. The local temperature component T_l of the overall temperature is calculated by taking a weighted average of the last 10 temperature values contributed by the last 10 processes that ran on the system. The weight of each value is the ratio of the time for which each process ran and the total time. The equation for calculating T_l is given below.

$$T_l = \sum_{i=1}^{10} T_i \frac{t_i}{t_{total}} \qquad \text{where} \qquad t_{total} = \sum_{k=1}^{10} t_i \tag{3}$$

T_i is the temperature value during schedule i, t_i is the duration of schedule i, and t_{total} is the total duration of the last 10 schedules. The linear weighted average used here is a good approximation because the last 10 values correspond to a very short time interval, which can be approximated using a linear equation. The overall global temperature T_g, which is a measure of the average energy stored in the processor, is calculated by feeding the aggregate counter values for all processes that ran on the system to our thermal model. The overall chip temperature is then given as:

$$T_{overall} = w_l T_l + w_g T_g \tag{4}$$

where $T_{overall}$ is the overall processor temperature, and w_l and w_g the weights corresponding to the local and global temperature components. Again, w_l is given a higher value than w_g to account for transient variations above the aggregate energy stored in the processor. In our experiments, we found that $w_{lp} = w_l = 0.7$ and $w_{gp} = w_g = 0.3$ gave the most accurate temperature estimates.

4.3 Proactive software-directed DTM

Thermal-aware process scheduling is an effective way to alleviate thermal emergencies with low performance impact. Using it proactively can help maintain the temperature below thermal emergency levels, and hence obviate the need for more aggressive DTM techniques which have a higher overhead.

The existing Linux scheduler tries to satisfy several objectives, such as fast response time, high throughput, fairness, etc., by ranking processes according to their priority and changing it dynamically. The lower the priority value, the higher the process rank. At a higher level, the entire CPU time is divided into epochs. Every process is also assigned a specific *timeslice*, which is a measure of the maximum CPU time that a process is allowed to run in an epoch. This scheduler, however, focuses solely on performance and is not energy- or thermal-aware. To make the scheduling policy thermal-aware, we dynamically *adjust process priorities* based on the following approach. When the overall temperature exceeds a pre-defined thermal trigger threshold T_{sw} for software DTM, we change the process priorities based on their estimated individual temperatures $T_{process}$. If $T_{process}$ for a particular process is higher than T_{sw}, its priority value is increased (rank is lowered) and vice-versa. We define *max_penalty* to be the maximum possible change in priority. The following pseudo-code shows our thermal-aware priority assignment policy.

```
for $process_i$ do
    $penalty\_ratio \leftarrow (T_{process_i} - T_{sw}) \ / \ (T_{max} - T_{sw})$
    if $penalty\_ratio > 1$ then
        $penalty\_ratio \leftarrow 1$
    $penalty \leftarrow penalty\_ratio \times max\_penalty$
    $priority \leftarrow old\_priority + penalty$
```

Using this algorithm, processes, whose $T_{process}$ exceeds the software trigger temperature T_{sw}, are qualified as "hot" and their priority value is increased (decreasing their rank) whereas processes, whose $T_{process}$ is less than T_{sw}, have their priorities decreased, thereby boosting their rank. The priority of a process remains unchanged if $T_{process}$ is equal to T_{sw}.

In addition to priority adjustment, we also implemented thermal-aware *timeslice adjustment*. In Linux, process timeslices are calculated based on their static priorities (nice values). Instead, we calculate timeslices based on the dynamic priority of a process. Since processes are already assigned priorities in a thermal-aware fashion, this makes our timeslice assignment thermal-aware, assigning shorter timeslices to "hot" processes and longer ones to "cold" processes.

4.4 Reactive hardware-directed DTM

While software-directed DTM has a low performance impact and can be application-aware, it may not be able to avoid thermal emergencies completely. In such cases, more aggressive hardware techniques, such as clock gating, can be employed to lower the overall temperature sharply by drastically decreasing the total power consumption.

Pentium 4 supports an elaborate clock gating mechanism which can be configured by software [1]. The OS can program the clock modulation MSR to throttle the processor clock in dicrete steps ranging from 12.5% to 87.5% throttling. When the overall temperature crosses a pre-defined hardware thermal trigger threshold T_{hw}, we use this feature of Pentium 4 reactively to manage the overall power consumption and hence temperature. This technique is engaged in three incremental steps, progressively increasing the clock throttling percentage from 12.5% to 25% and finally to 50% with increasing temperature values. The software overhead involved in engaging and disengaging this technique is minimal because it only entails writing values to a specific MSR which can be done in a few processor clock cycles. The low overhead allows an extremely **fine-grained control** where clock gating is engaged or disengaged (based on the overall temperature) at the granularity of a context switch, thereby not using it for longer than is needed, and hence minimizing its performance impact. While in this work, we use hardware DTM only in the event of thermal emergencies, proactive use of both hardware and software DTM will be studied in the future.

The Pentium 4 processor supports another on-die thermal diode [2] which is different from the one we used while developing our thermal model. This diode acts as a catastrophic shutdown detector, triggering automatic processor shut down when the temperature becomes alarmingly high (around 135°C), to prevent burn-out. This feature complements our proposed DTM technique and acts as the ultimate fail-safe mechanism.

4.5 Changes for SMT

The Pentium 4 processor used in our setup supports two logical processors per physical processor in an SMT environment [12]. Most of the performance counters in Pentium 4 are thread-specific (TS) and can be configured to count events occurring on only specific logical processors. However, some counters, which are thread-independent (TI), cannot be configured to obtain counts for individual logical processors [1]. Moreover, there is only one set of counters for every hardware event which makes it impossible to simultaneously count events for two processes running on different logical processors. Therefore, for TI events, we assume equal counts for both logical processors while TS counters are time-multiplexed between the two logical processors. Because of these limitations, the error in estimating the temperature $T_{process}$ for each process, as well as $T_{overall}$, is increased as compared to the uniprocessor case. Note though that processes running on both logical processors contribute to $T_{overall}$ because they actually run on the same physical processor.

5. EVALUATION

In this section, we evaluate the performance of our proposed hybrid DTM policy using a number of integer and floating-point benchmarks from the SPEC2000 benchmark suite. These benchmarks were compiled using gcc-3.4.2 using "-O3 -fomit-frame-pointer" flags. The two major metrics to consider while evaluating a DTM strategy are: effectiveness in eliminating all thermal emergencies to ensure safe on-line operation and the execution time overhead. We analyze HybDTM based on these two metrics. For comparison, we consider the following alternative DTM strategies:

- **Software-based DTM (SDTM):** SDTM uses the two proposed software mechanisms, thermal-aware priority management and timeslice management, to manage overall temperature. This scheme helps determine if using only software-based mechanisms can successfully avert all thermal emergencies. We also analyze the performance impact of using this scheme.
- **Hardware-based DTM (HDTM):** This strategy uses a fine-grained clock gating approach to manage temperature. In this technique, as temperature rises, the clock gating ratio is increased incrementally from 0 to 12.5%, 25% and finally 50%. This scheme helps identify the performance impact of using only hardware-based DTM.

In our experiments, we set T_{max}, which is the maximum allowable processor temperature depending on the thermal budget and cooling solution, to 65°C. We set T_{sw}, the thermal trigger threshold for proactively invoking software DTM, to 60°C and T_{hw}, the threshold for invoking hardware DTM, to 62°C. The processor clock is throttled in incremental steps when the temperature reaches T_{hw}, i.e., 12.5% throttling at T_{hw}, 25% at T_{hw}+1 and 50% at T_{hw}+2. The temperature values chosen were low because of the efficient cooling mechanism in our system.

5.1 Uniprocessor results

We analyze the effectiveness of HybDTM for the uniprocessor configuration by running several SPEC2000 benchmarks and comparing it with the alternative DTM strategies proposed above. To analyze the effect of proactive software techniques in HybDTM, we also run *memory*, a moderately energy-efficient microbenchmark, in parallel with SPEC2000 benchmarks. The *memory* microbenchmark traverses a large integer array in a distributed fashion and stores values at different locations within the array. Hence, it uses the processor's address generation unit but overall is designed to be a memory system intensive benchmark. The peak processor temperature reached when running the *memory* microbenchmark alone is 55°C.

Thermal management: Table 1 shows the peak temperature values for *gzip*, *vpr* and *twolf* from the SPECint benchmark suite and *wupwise* and *applu* from the SPECfp suite for both HybDTM and without any DTM. First, we can see that HybDTM is successfully able to manage the peak temperature below the maximum allowed temperature, T_{max}, and hence guarantee thermal safety in all cases. For *gzip*, *vpr* and *applu*, the difference between the observed peak temperature and T_{max} is less than or equal to 1°C, which implies that the peak temperature exceeds the hardware thermal trigger temperature T_{hw}. Hence, both software priority/timeslice management as well as reactive hardware clock gating were needed to guarantee thermal safety. Moreover, all levels of clock gating were active. For *twolf*, the peak temperature is below T_{hw}, which implies that proactive software mechanisms were sufficient to avert thermal emergencies. For *wupwise*, the peak temperature is 2°C below T_{max}, hence both proactive and reactive techniques were needed to manage temperature, but only two out of the three clock gating levels were active.

Fig. 4 shows the average CPU utilization for the different benchmarks. It can be seen that in the absence of DTM, both *memory* and the SPEC benchmarks utilize around 50% of the CPU. However, in the case of HybDTM, thermal-aware scheduling reduces the CPU utilization of the relatively "hot" SPEC benchmark, while increasing the CPU utilization of *memory*, as it taxes the CPU less.

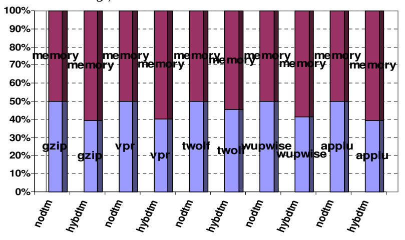

Figure 4: Relative CPU utilization

Comparative study: To isolate the impact of both the hybrid nature of HybDTM and its fine-grained policy, we compare the effectiveness and performance impact of HybDTM with both SDTM and HDTM.

In our experiments, we found that SDTM alone was unable to manage the peak temperature below T_{max}. Hence, the use of a software-only approach to guarantee thermal safety is insufficient and hardware mechanisms are needed to deal with thermal emergencies. HDTM, on the other hand, was able to successfully guarantee thermal safety.

Fig. 5 shows the performance impact in terms of relative execution time overhead of HybDTM and HDTM. The execution times are normalized to the time taken by a single run of the benchmark in the absence of any DTM policy. We see that HybDTM has a lower performance impact as compared to HDTM in all cases. Note that the execution time overhead shown is that of the SPEC benchmark with *memory* running in the background. The average execution time overhead for HybDTM compared to the case when no DTM technique is employed is 9.9%, with the maximum being 16.3%. The corresponding overhead for HDTM is 20.4% (29.5% maximum). The average improvement of HybDTM over HDTM is 9.9%, with the maximum being 16.9%, which occurs in the case of *applu*. The performance impact for *twolf* is low, because as mentioned earlier, only low overhead proactive software techniques were sufficient to guarantee thermal safety.

Analysis: Fig. 6 shows the variation in the processor temperature with time for *gzip* running along with *memory*. From the figure, the effects of hardware vs. software DTM can be clearly seen. Due to the proactive use of priority/timeslice adjustment, the rank assigned to the "hot" *gzip* benchmark is low and, hence, the initial temperature rise is not very steep. Moreover, as the temperature rises above T_{hw} and clock gating is enabled, the temperature drops sharply. The subsequent rise is again very gradual

Table 1: Peak temperature for SPEC2000 benchmarks

Uniprocessor configuration	gzip+memory	vpr+memory	twolf+memory	wupwise+memory	applu+memory
Without DTM	69.5	68.5	66.5	67.5	68.5
HybDTM	64.5	64.0	61.5	63.0	64.5
SMT configuration	twolf+wupwise	gzip+vpr	vpr+equake	gcc+swim	art+equake
Without DTM	73.0	75.0	72.5	71.5	71.5
HybDTM	63.5	64.5	64.5	64.0	64.5

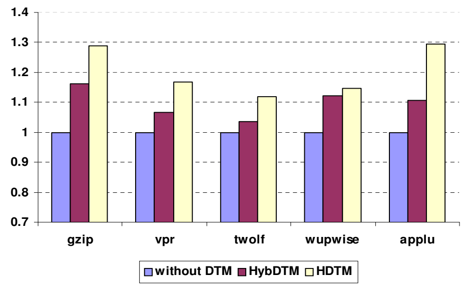

Figure 5: Relative execution time for different DTM policies

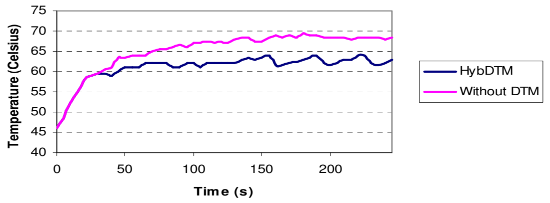

Figure 6: Temperature variation of gzip+memory

because of the lower rank of *gzip* compared to the relatively "cold" *memory*. Therefore, clock gating is not enabled very often. Furthermore, as explained in Section 4.4, we use an extremely fine-grained control for clock gating by engaging/disengaging it at the granularity of every context switch. This ensures that the clock is throttled only as long as needed, thereby minimizing its performance impact. Hence, by taking advantage of low overhead software mechanisms and using hardware mechanisms as a fail-safe backup, HybDTM is able to minimize the performance impact while ensuring thermal safety.

5.2 SMT results

Table 1 shows the peak temperature of several combinations of SPECint and SPECfp benchmarks running simultaneously in a multi-threaded environment. The trigger thresholds, T_{sw} and T_{hw}, as well as the maximum allowed temperature, T_{max}, are the same as in the case of the uniprocessor configuration. Again, it can be seen that the peak temperature with HybDTM is always below T_{max}, which implies that HybDTM is able to guarantee safe on-line operation. In all cases, hardware clock gating was required in addition to priority/timeslice management to manage the overall temperature.

Fig. 7 shows the normalized execution times for the different benchmark pairs. Again, HybDTM performs better than HDTM in all cases. However, the proactive software technique is not very effective here because all SPEC benchmarks exhibit high thermal contributions to the overall temperature and hence are "hot". Therefore, priority/timeslice management lowers the rank of both benchmarks in each pair, although by different amounts. Due to this, the relative improvement in performance impact of HybDTM over HDTM is not high and clock gating is used frequently in both cases to manage the peak temperature.

6. CONCLUSION

As microprocessor power densities and temperatures increase, the key challenge in DTM research is to guarantee thermal safety while minimizing the performance impact. In this paper, we presented a low overhead regression-based processor thermal model that directly uses hardware performance counters to accurately characterize the thermal impact of individual applications at run-time while at

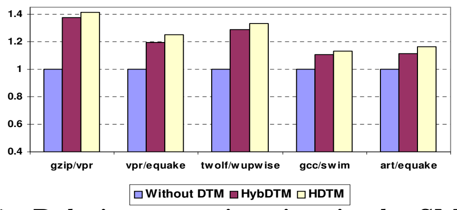

Figure 7: Relative execution time in the SMT case

the same time taking into account system-level thermal interactions between different components like the processor and memory. Using this model, we proposed and evaluated HybDTM, a novel DTM mechanism which uses a hybrid of hardware and software techniques to manage the overall temperature while minimizing the performance impact. We hope to extend this work by developing a framework for more fine-grained DTM, considering the interactions between a number of hardware and software DTM techniques, such as, DVS, fetch toggling, task migration, etc., in high-performance systems. We see this work forming the foundation for low overhead system-level DTM solutions.

7. REFERENCES

[1] *IA-32 Intel architecture software developer's manual, Vol. 3: System programming guide.* `http://developer.intel.com/design/pentium4/manuals/245472.htm`

[2] *Intel Pentium 4 processor in the 478-pin package thermal design guidelines.* `http://developer.intel.com/design/pentium4/guides/249889.htm`.

[3] *Mobile Intel Pentium 4 processor – M datasaheet.* `http://www.intel.com`.

[4] J. Baek *et al.*, "Thermal characterization of high speed DDR devices in system environments," in *Proc. Ninth Annual IEEE Semiconductor Thermal Measurement and Management Symp.*, Mar. 2003.

[5] F. Bellosa *et al.*, "Event-driven energy accounting for dynamic thermal management," in *Proc. Wkshp. Compilers and Operating Systems for Low Power*, Sept. 2003.

[6] D. Brooks *et al.*, "Dynamic thermal management for high-performance microprocessors," in *Proc. Int. Symp. High Performance Computer Architecture*, Jan. 2001, pp. 171–182.

[7] X. Fan *et al.*, "Memory controller policies for DRAM power management," in *Proc. IEEE Symp. Low Power Electronics*, Aug. 2001.

[8] G. Hilton *et al.*, "The microarchitecture of the Pentium 4 processor," *Intel Technology Journal*, Feb. 2001.

[9] M. Huang *et al.*, "A framework for dynamic energy efficiency and temperature management," in *Proc. Int. Symp. Microarchitecture*, Dec. 2000, pp. 202–213.

[10] C. Isci *et al.*, "Run-time power monitoring in high-end processors: Methodology and empirical data," in *Proc. Int. Symp. Microarchitecture*, Dec. 2003.

[11] J. Janzen, "Calculating memory system power for DDR SDRAM," *Designline*, vol. 10, no. 2, 2001.

[12] D. Koufaty *et al.*, "Hyperthreading technology in the netburst microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 56–65, Mar. 2003.

[13] K.-J. Lee *et al.*, "Using performance counters for runtime temperature sensing in high-performance processors," in *Proc. Wkshp. High-Performance Power-Aware Computing*, Apr. 2005.

[14] L. Shang, L.-S. Peh, A. Kumar and N. K. Jha, "Thermal modeling, characterization and management of on-chip networks," in *Proc. Int. Symp. Microarchitecture*, Dec. 2004.

[15] K. Skadron, "Hybrid architectural dynamic thermal management," in *Proc. Design Automation and Test in Europe Conf.*, Feb. 2004, pp. 10–15.

[16] K. Skadron *et al.*, "Temperature-aware microarchitecture," in *Proc. Int. Symp. Computer Architecture*, June 2003, pp. 1–12.

[17] B. Sprunt, "Pentium 4 performance-monitoring features," *IEEE Micro*, vol. 22, no. 4, pp. 72–82, Jul./Aug. 2002.

[18] A. Weissel *et al.*, "Dynamic thermal management for distributed systems," in *Proc. Wkshp. Temperature-Aware Computer Systems*, June 2004.