Predicting Thermal Behavior for Temperature Management in Time-Critical Multicore Systems

Buyoung Yun and Kang G. Shin
EECS Department, University of Michigan
Ann Arbor, MI 48109-2121
{*buyoung,kgshin*}*@eecs.umich.edu*

Shige Wang
General Motors Global R&D
Warren, MI 48090
shige.wang@gm.com

***Abstract*—Multi-core System-on-Chip (SoC) has become a popular execution platform for many embedded real-time systems that require high performance and low power-consumption. High temperature is known to accelerate the failure of deep submicron chips. To prevent such accelerated failures due to chip overheating, various *thermal-aware scheduling* (TAS) algorithms and *dynamic thermal management* (DTM) have been proposed and applied to mission/safety-critical applications. To control on-chip temperature more effectively, it is necessary to predict the thermal dynamics of a multi-core chip in real time and trigger appropriate power/temperature management before overheating the chip. However, due to dynamically-changing runtime environments, it is very difficult to estimate the chip temperature on-the-fly.**

In this paper, we propose models of efficiently estimating multi-core chip temperature while accounting for the system dynamics in real time. Based on these models, we design a proactive *peak temperature manager* (PTM) which periodically estimates future core temperature and triggers proper DTMs on the estimated-to-be-overheated cores for their cooling without violating applications timing constraints. Our in-depth evaluation based on the HotSpot thermal simulator has shown that the proposed method can predict the occurrence of peak temperature in a core with 90–98% accuracy, and using the estimated thermal model of a multi-core chip, PTM can effectively keep core temperature below a given threshold without violating any timing constraint.

I. Introduction

Use of multi-core chips to meet the non-functional requirements of embedded real-time systems has been drawing significant interest due mainly to their potential for high performance and reliability at low cost. The state-of-the-art CMOS scaling-down technology enables a chip to contain more processing units on a single die to achieve higher performance. However, as CMOS process technology continues to scale down into the nano-meter, the power density on a chip continues to rise, generating more heat and thermal hotspots on the chip than ever before. High temperature is known to accelerate the degradation/failure of electronic circuits. High on-chip temperature also affects the functional and timing correctness of the chip's operation [1], and will eventually cause irrecoverable failures. Hence, proper temperature management on a multi-core chip is crucial to reduction of the risk of failure resulting from thermal hotspots, and satisfaction of the non-functional requirements of embedded real-time systems.

To address the unreliability caused by high chip temperature, various thermal-aware scheduling (TAS) algorithms have been used for safety/mission-critical applications running on a uni/multi-core processor. Of these, the TAS algorithms using dynamic voltage and frequency scaling (DVFS) have been studied extensively [2], [3]. Assuming that the chip thermal model is estimated to be linear, and also available at system design time, both the timing and thermal constraints are guaranteed by those methods at design time. However, since the workload and surrounding environment greatly affect the chip temperature at runtime, it is very difficult to obtain an accurate linear thermal model at design time without considering the runtime information.

To circumvent this difficulty in obtaining an accurate chip thermal model, power/temperature management can be triggered *reactively* at runtime, i.e., the system slows down a core's speed or turns it off for a certain period of time when the core temperature exceeds a given threshold. However, such reactive methods cannot prevent overshooting of the chip temperature. To lower the chance of violating the peak temperature constraint, we need frequent monitoring of core temperature. All of these contribute to inefficient thermal management and increase the runtime overhead. For efficient thermal management, several proactive schemes based on linear autoregressive (AR) models have been proposed [4], [5]. Despite the simplicity of the linear AR model, its use of a core's past temperature traces makes it difficult to use for the case of running multiple tasks on the core. In particular, when the amount of heat generated by the executing tasks varies significantly, a core's temperature prediction based solely on its past temperature traces becomes inaccurate as they do not accurately reflect the core's future thermal behavior.

To remedy/mitigate the above problems, we first design a runtime peak temperature manager (PTM), which is invoked periodically to *proactively* avoid overheating a multi-core chip. We assume that a given set of independent periodic tasks are partitioned and assigned to cores so as to meet the task timing constraints by a given scheduling algorithm. The proposed PTM is then invoked periodically at runtime to operate on the given task partition and along with the given scheduling algorithm. Depending on whether or not each core has its own separate power domain, the PTM executes locally or globally.

To predict the future thermal behavior of a core at the time of PTM invocation, we propose a simple but effective runtime method for estimating the thermal model of each core. For a given time window, each core is categorized as "overheated" or "not-overheated" using a temperature threshold, which is a design parameter. We formulate the chip-temperature prediction as a binary classification problem, and apply the Support Vector Machine (SVM), a popular machine learning technique. Thus, each core has its own thermal model based on the SVM, and the PTM keeps the thermal models up-to-date using runtime information. To determine whether or not a core will be overheated before the next PTM invocation, we use the thermal profiles (obtained off-line) of tasks scheduled to execute during the next PTM period, as well as the currently-measured core temperatures. Then, depending on the underlying hardware support, the PTM can apply DVFS or power-gating locally or globally. Based on the proposed PTM and the thermal prediction models, we incorporate such temperature management schemes to avoid overheating without violating any timing constraint. The performance of PTM with temperature-management schemes is evaluated via realistic simulation and analysis in terms of effectiveness, time-complexity, and limitations.

The rest of this paper is organized as follows. Section II states the assumptions and describes the system models. Section III presents the proposed PTM and formulates the thermal model estimation problem. Section IV describes a method for constructing temperature prediction models using SVM for its online use. Section V presents various temperature-management schemes that the proposed PTM can use. Section VI evaluates the performance of the PTM using extensive simulations. Section VII discusses the related work, followed by the concluding remarks in Section VIII.

II. System Models

To design the runtime PTM for multi-core chips, we need to define a set of models, including the processor, the task and the scheduler models.

A. Processor and Chip Models

A single-chip system $\mathbf{S}$ with M cores is represented by $\mathbf{S} = \{core_i\}_{i=1}^{M}$. All cores may be identical or heterogeneous.

To measure the core temperature at runtime, we assume that on-chip temperature sensors are deployed on a multi-core chip. In practice, it is difficult to obtain accurate core temperature values due to measurement noise and sensor placement. It was reported that the error of temperature measurement is around $1°C$ and the accuracy is further improved with more sensors [6]. In this paper, we assume that a sufficient number of accurate temperature sensors have been deployed on a chip to minimize the measurement errors, and the maximum hotspot temperature in each core can be acquired from the temperature sensors on the core.

Also, the thermal dynamics of a core are coupled with the thermal activities of its neighbor cores in a multi-core chip. Therefore, the knowledge of the chip's physical layout of cores is essential for the estimation of each core's temperature. To this end, the neighbor cores of $core_i$ in $\mathbf{S}$ are assumed known. For example, the neighbor cores of $core_2$ in Fig. 1 are $\{core_1, core_3, core_5, core_6, core_7\}$.

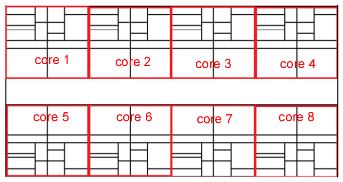


Figure 1. Layout of 8 cores on a chip.

B. Thermal Dynamics on a Multi-core Chip

The thermal behavior of a multi-core chip can be described by its equivalent thermal circuits, where each component (such as cores, heat spreads, heat sinks, interconnectors, etc.) is considered as a thermal node [3], [7]. By assuming that thermal dynamics of a chip is a linear and time-invariant (LTI) system and the number of thermal nodes on a chip except cores is H, the thermal dynamics of a chip can be described as a discrete-time linear state equation as [8]:

$$\mathbf{x}(n) = \mathbf{A}\mathbf{x}(n-1) + \mathbf{B}\mathbf{u}(n-1) \tag{1}$$

where $\mathbf{x}(n-1)$ and $\mathbf{u}(n-1)$ are $(M+H)$-element vectors representing the temperatures and power consumption at the thermal nodes at time $\mathbf{t} = n \cdot \Delta t$. For convenience, we assume that $core_i$'s temperature and power dissipation is the i-th element in the corresponding vector. Also, $\mathbf{A}$ and $\mathbf{B}$ are $(M+H) \times (M+H)$ matrices. The PTM does not require any prior knowledge of $\mathbf{A}$ and $\mathbf{B}$ at both system design time and runtime.

C. Task Model

The task system under consideration is composed of N independent periodic tasks $\mathbf{T} = \{\tau_i\}_{i=1}^{N}$. We assume that each task τ_i is executed only on its designated core, and let T_j denote a set of tasks designated on $core_j$. A task τ_i is modeled as $\tau_i = (e_i, \hat{e}_i, p_i, \Theta_i)$ where e_i is the worst-case execution time, $\hat{e}_i$ the average-case execution time, p_i the period of τ_i and Θ_i an additional parameter to represent the amount of heat generation during the task execution. Task τ_i is invoked every p_i and each such invocation is called a job. We assume that the relative deadline of a task is equal to its period, meaning that each job must be completed before the next job of the same task is released. In a case where a multi-core chip can change its operating frequency using

DVFS, let e_i and $\hat{e}_i$ denote the worst-case and average-case execution times of τ_i when the operating frequency is set to the maximum speed, f_{max}.

In addition, the core temperature depends on the power consumption of the tasks executing on it. Different tasks consume different amounts of power. For accurate prediction of the cores' thermal behavior, we thus need a metric to identify the relative amount of heat generated during each task execution. For this purpose, we define an additional parameter, Θ_i, called the *thermal profile* of τ_i. For example, the average power consumption of τ_i's execution can be used as Θ_i. However, because on-chip power sensors are not widely used in modern microprocessors, a steady-state temperature during the execution of a task is used to define its thermal profile in this paper.

Let $\hat{P}_i$ denote the average power consumption of τ_i's execution. We assume that τ_i is executed repeatedly on its designated core, $core_k$ in $\mathbf{S}$, for a sufficiently long period of time while other cores are in sleep mode. Using Eq. (1), the cores' steady-state temperatures are expressed as:

$$\mathbf{x}(\infty) = \mathbf{A}\mathbf{x}(\infty) + \mathbf{B}\tilde{\mathbf{u}}$$

where $\tilde{\mathbf{u}}_{j,1} = \hat{P}_i$ for $j = k$ and $\tilde{\mathbf{u}}_{j,1} = \epsilon(\simeq 0)$ otherwise. Thus, $\hat{P}_i \simeq \Theta_i/\mathbf{D}_{k,k}$ where $\mathbf{D} = (I - A)^{-1}\mathbf{B}$, meaning that $\Theta_i/\Theta_j = \hat{P}_i/\hat{P}_j$ for any τ_i and τ_j in T_k.

When the task set $\mathbf{T}$ is given at system design time, the steady-state temperature of each task can be obtained off-line by recording the on-chip temperature sensor readings while repeatedly executing the task on its designated core without any DTM mechanism for a sufficiently long period of time. For this, the inputs of each task should be known at design time, because the amount of power consumption varies with its task execution path. This process is akin to the empirical estimation of task execution time.

All of the thus-obtained tasks' thermal profiles are used for classification and estimation of the future core temperatures at runtime. Since a thermal profile is used to differentiate the relative amount of heat generated by a task from others', the operational environment at design time, such as room temperature, need not be the same as that at runtime. Using the same processor model in the HotSpot thermal simulator, we obtained the steady-state temperatures of MiBench programs as given in Table I.

Table I
ESTIMATED STEADY-STATE TEMPERATURE OF TESTBENCH PROGRAMS ($^{\circ}C$)

bitcnt	fft	gsm	dijkstra	rijndael	sha	qsort	susan
89.31	76.18	82.94	90.97	75.23	89.9	80.44	91.0

D. Scheduler and PTM Models

We assume that each core has its own local scheduler. Before the system starts, the tasks in $\mathbf{T}$ are assigned to the cores in $\mathbf{S}$ so as to satisfy the schedulability condition on every core with the assigned tasks, assuming that the operating frequency of all cores are set to f_{max}.

At runtime, the local scheduler and the PTM cooperate with each other. First, the PTM is invoked every p_t, which is chosen at the system design time. At the beginning of every p_t, the PTM interrupts its corresponding cores to monitor their temperature, and estimates whether or not a core will be overheated during the next p_t using the given temperature threshold and the temperature prediction model. If a core is expected to be overheated during the next p_t, the PTM applies DTM to cool it. Here, we choose two DTM schemes, DVFS and power-gating, and discuss how to incorporate them into PTM in Section V. The PTM operates either locally or globally. Depending on the underlying hardware implementation, the power domain of each is separated from others', or is shared by all cores in $\mathbf{S}$. If each core has its own power domain, each core has its own PTM, but when it shares the power domain with all other cores, there exists one PTM in $\mathbf{S}$, and executed on a randomly-selected core in $\mathbf{S}$.

The local scheduler on a core executes the released jobs under the earliest-deadline-first (EDF) scheduling policy. The following variables on $core_l$ are updated by its local scheduler at runtime:

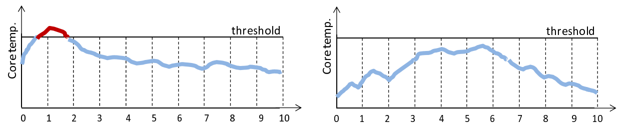
1) Completed portion c_i of the job of $\tau_i \in T_l$, which is currently running on the core. Given τ_i's execution time e_i and $\hat{e}_i$, it is used to calculate the average-case remaining execution time of jobs, which is then used to estimate the core's future thermal behavior. The completed job portions are also used by the PTM to check the available time slack on the core when applying the proper DTM.
2) Event of peak-temperature occurrence on $core_l$ during every p_t interval. Depending on the underlying hardware support, peak temperature reaching the peak temperature during each p_t can be alerted by an interrupt [9], or obtained by sampling the core temperature.

The local scheduler keeps and updates these variables at the next PTM invocation time.

III. PROBLEM STATEMENT

Our PTM is designed with two constraints in mind: (1) meeting all timing constraints of the given task set, and (2) maintaining the peak temperature of the cores below a given threshold throughout the system operation. It may not be feasible to meet both constraints in certain cases due to the limit of hardware resource capacity and the tightness of task timing constraints. In such a case, our algorithm treats the constraint (2) as soft, but treats the constraint (1) as hard.

To effectively control the temperature on a multi-core chip, the PTM relies on the chip temperature prediction models to avoid overheating any part of the chip. As the PTM is invoked every p_t, it should be able to predict the thermal behavior of a core(s) during the next PTM interval upon its invocation.



(a) overheated (b) not overheated

Figure 2. Classification of the overheated core with $p_t = 10$ and $p_s = 1$.

Upon invocation of the PTM at $\mathbf{t} = t$, a core is said to be "overheated" if its temperature is estimated higher than the given threshold at some time during $[t, t + p_t)$ when the core clock frequency is set to f_{max}. Fig. 2 illustrates how a core is classified as overheated. We assume that p_s is given as a design parameter where p_s is a divisor of p_t and the power dissipation on each core is almost constant during p_s interval. Also, let $t = n \cdot \Delta t$ and $p_s = x \cdot \Delta t$. Using Eq. (1), the core temperature at time $\mathbf{t} = t + k \cdot p_s$ is derived as:

$$\mathbf{x}(n + kx) = \mathbf{A}^{kx}\mathbf{x}(n) + \sum_{j=0}^{k-1}\sum_{l=1}^{x} \mathbf{A}^{kx+jx-l}\mathbf{B}\tilde{\mathbf{u}}(j)$$

where $\tilde{\mathbf{u}}(l)$ is the expected power vector at $\mathbf{t} = t + l \cdot p_s$. Since the power dissipation on each core is assumed not to vary during p_s, if the core's temperature is below the threshold at $\mathbf{t} = t + k \cdot p_s$ $(0 \le k \le p_t/p_s)$, it is guaranteed not to be overheated during $[t, t+p_t)$. For simplicity, $core_i$'s temperature at $\mathbf{t} = t + k \cdot p_s$ is expressed as:

$$\mathbf{x}_{i,1}(n + kx) = g_{i,k}(\{\mathbf{x}_{j,1}(n)\}_{j \in N_i}, \{\tilde{\mathbf{u}}_{i,1}(j)\}_{j=0}^{k-1}) + \Delta_{i,k} \tag{2}$$

where $g_{i,k}(\cdot)$ is a linear function of its inputs, N_i is the set of $core_i$'s neighbors and $\Delta_{i,k}$ the sum of other remaining terms, which are not dependent on $\{\mathbf{x}_{j,1}(n)\}_{j \in N_i}$ and $\{\tilde{\mathbf{u}}_{i,1}(j)\}_{j=0}^{k-1}$. When p_t is chosen smaller than the thermal time constant of a chip, the first term in Eq. (2) becomes dominant. Also, because the thermal profile of a task is proportional to its average power consumption, $\mathbf{x}_{i,1}(n+kx)$ can be approximated as a linear function of $\{\mathbf{x}_{j,1}(n)\}_{j \in N_i}$ and the thermal profiles of tasks executing at time $\mathbf{t} = t+jp_s$ $(0 \le j \le k - 1)$. Thus, for the given temperature threshold, Θ_{thres}, overheating $core_i$ can be predicted using:

$$\prod_{k=1}^{p_t/p_s} u_{-1}(\Theta_{thres} - \mathbf{x}_{i,1}(n + kx)) \tag{3}$$

where $u_{-1}(\cdot)$ is the unit step function. However, since $\mathbf{A}$ and $\mathbf{B}$ are unknown, and the temperature of some thermal nodes (e.g., heat sinks and heat spreads) on a chip cannot be measured, Eq. (3) cannot be used directly in the PTM. Thus, using the chip model, the task model, and the scheduler model, the temperature-prediction problem in our PTM can be defined formally as follows.

> <u>Given</u> a set of periodic independent tasks $\mathbf{T}$, a multi-core system $\mathbf{S}$, and a temperature threshold Θ_{thres}, <u>find</u> a runtime temperature-prediction model for each $core_i$ in $\mathbf{S}$ that generates the same results of Eq. (3) during $[t,\ t+p_t)$ when the PTM is executed at time $\mathbf{t} = t$.

To solve this problem, we adopt the SVM-based thermal prediction model as detailed in Section IV. When a core is predicted to be overheated, the PTM applies its available DTM to the core as follows.

- *Adjust the core speed* (DVFS): The core's speed is set to the lowest clock frequency below f_{max} without violating the timing constraints of tasks in $\mathbf{T}$. Otherwise, the core operates at f_{max}.
- *Perform power-gating:* The PTM calculates the maximum available time slack during $[t,\ t+p_t)$ on the core and turns it off during the time slack without violating the timing constraints of tasks in $\mathbf{T}$.

Each DTM scheme can be applied globally or locally. Also, when the core is slowed down or turned off, all the assigned tasks should not miss their deadlines: this will be detailed in Section V.

IV. Thermal Prediction Model

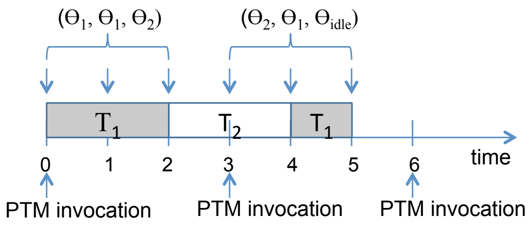
The temperature of a core changes dynamically during runtime and is affected by many factors, such as the workload, the temperature of its neighbor blocks, the material, the room temperature and the cooling mechanisms on the chip. As a result, the PTM requires a temperature-prediction model for each core to estimate its temperature at runtime. Due to the high complexity of the underlying thermal dynamics of a multi-core chip, we cannot determine a complex thermal model analytically at runtime. Thus, we adopt a binary classification method and the SVM-based model to predict whether or not the core will be overheated at each PTM invocation. Since the cores on a chip are placed at different locations, each core has its own prediction model. So, we need to obtain M thermal prediction models for a multi-core chip $\mathbf{S}$ for use at runtime.

The SVM is a well-known supervised machine learning method in which each feature vector contains a set of numerical features, which are deemed important in representing the target system. The SVM takes a set of feature vectors as input, along with their corresponding labels, which are used for training the prediction models. Then, the thus-learned models are used to predict the label of future input data. In this paper, a core's thermal behavior during $[t, t + p_t)$ is represented as a feature vector, and each feature vector is labeled as "overheated" or "not overheated." To guarantee high prediction accuracy with minimum runtime overhead, it is essential to select a good feature vector. From Eqs. (2) and (3), one can see that a core's thermal behavior during $[t, t + p_t)$ is affected mainly by the following parameters attributing to the feature vector during that interval.

1) The measured temperatures of the core and its neighbors at $\mathbf{t} = t$. Because all cores are on the same die, the core's future temperature is affected by the temperature of all cores. However, for runtime efficiency, only the

measured temperatures of its neighbors are included in the feature vector.

2) The average thermal profiles of tasks scheduled to execute on the core at each p_s time interval. For example, two tasks $\tau_1 = (2, 2, 4, \Theta_1)$ and $\tau_2 = (2, 2, 7, \Theta_2)$ are assigned on the core and $p_s = 1$, $p_t = 3$. We assume that the first jobs of τ_1 and τ_2 consume their worst-case execution times, but the second job of τ_1 only consumes 1 time unit. As shown in Fig. 3, $\{\Theta_1, \Theta_1, \Theta_2\}$ are the thermal profiles of tasks executing during $[0, 3)$ in this case. However, due to the variation of jobs' actual execution times, there exist other cases with different thermal profiles of tasks during the same interval. To account for this, the average thermal profiles among all possible thermal profiles are used in the feature vector. In this example, because the average response times of the first jobs of τ_1 and τ_2 are 2 and 4, respectively, $\{(\Theta_1 + \Theta_2)/2, (\Theta_1 + \Theta_2)/2, \Theta_2\}$ are included in the feature vector during $[0, 3)$



$(\Theta_1, \Theta_1, \Theta_2)$
$(\Theta_2, \Theta_1, \Theta_{idle})$
T1
T2
T1
0
1
2
3
4
5
6
time
PTM invocation
PTM invocation
PTM invocation

Figure 3. An example of tasks' thermal profiles.

Once the feature vector is obtained for $[t, t + p_t)$, its class label can be assigned after time $\mathbf{t} = t + p_t$ from the local scheduler as described in Section II-D. Using this information, the SVM-based thermal model is initialized, and trained online. We first present the method for deriving the average thermal profiles of tasks included in the feature vector, and then describe the methods for initializing the model parameters in SVM, and training the SVM-based thermal model online.

A. Feature Vector

To predict whether the peak temperature will occur in a core during $[t, t + p_t)$ at the PTM invocation time $\mathbf{t} = t$, the PTM should construct a feature vector for this interval. It includes the measured temperatures of the core and the neighbor cores at time $\mathbf{t} = t$, which can be easily obtained from the on-chip temperature sensors, but requires the expected thermal profiles of the tasks scheduled to execute on the core during this interval. However, due to the unpredictability of the actual execution time of the tasks, the PTM should consider the average-case scenario from the thermal perspective.

Thus, to estimate the average-case thermal profiles of the tasks during $[t, t + p_t)$, let S_i be a set of integers between $\lfloor t/p_i \rfloor$ and $\lfloor (t+p_t)/p_i \rfloor$. Then, for $\forall k \in S_i$, the k-th job of τ_i is either a job stored in the job queue at $\mathbf{t} = t$ or a job which will be released during $[t, t+p_t)$. Let $R_{i,k}$ denote the average-case response time of the k-th job of τ_i, then the average-case thermal profile of a task at $\mathbf{t} = t + jp_s$ $(0 \le j < p_t/p_s)$ on $core_l$ will be obtained as:

$$\frac{1}{\mathcal{N}_{t,j}} \sum_{\forall \tau_i \in T_l} f_i(t + jp_s) \tag{4}$$

where $\mathcal{N}_{t,j}$ is the number of tasks in T_l such that $f_i(t + jp_s) \neq 0$, and

$$f_i(\mathbf{x}) = \begin{cases} \Theta_i & kp_i \le \mathbf{x} < R_{i,k} \text{ for any } k \in S_i \\ 0 & \text{otherwise,} \end{cases}$$

Thus, to construct a feature vector and predict the thermal behavior of a core during the interval $[t, t+p_t)$, we need to obtain $R_{i,k}$ for $\forall k \in S_i$ and $\forall \tau_i \in T_l$. For this, $W_i(\mathbf{t},\ \mathbf{x})$ is defined as a function of the average-case remaining workload in the job queue on the core at time $\mathbf{t}$ which should be completed before $\mathbf{x} \cdot p_i$. Using this definition, $kp_i + W_i(\max(t,\ kp_i),\ k+1)$ is the average-case response time of the k-th job of τ_i without considering the blocking time by the higher-priority jobs of other tasks released during $(kp_i, (k+1)p_i]$. To obtain $W_i(\max(t,\ kp_i),\ k+1)$, $W_i(t,\ k+1)$ on $core_l$ is initialized using the recent runtime information from the local scheduler as:

$$W_i(t,\ k+1) = \sum_{\substack{\tau_j \in T_l \\ \lfloor t/p_j \rfloor (p_j+1) \le (k+1)p_i}} h(\hat{e}_j - c_j, e_j - c_j).$$

where $h(\mathbf{a}, \mathbf{b}) = b$ if $a \le 0$, and a otherwise. $W_i(kp_i,\ k+1)$ is then obtained using the iterative method as:

$$x' \leftarrow x$$

$$x \leftarrow \min_{\forall \tau_j \in T_l} (\lfloor \frac{x}{p_j} \rfloor + 1)p_j,\ \ \Delta = x - x'$$

$$W_i(x,\ k+1) \leftarrow u_{-1}(W_i(x',\ k+1) - \Delta) + \sum_{\tau_j \in N_l(x)} \hat{e}_j$$

where $u_{-1}(\mathbf{x}) = \max(\mathbf{x}, 0)$ and $N_l(x)$ is the set of tasks on $core_l$ whose new jobs are released at $\mathbf{t} = x$ and $x + p_j \le (k+1)p_i$ for $\forall \tau_j \in N_l(x)$. The above iteration stops when $x = \max(t,\ kp_i)$.

$R_{i,k}$ is initialized as $kp_i + W_i(kp_i,\ k+1)$. By applying the iterative method again, $R_{i,k}$ can be obtained as:

$$R'_{i,k} \leftarrow R_{i,k}$$

$$R_{i,k} \leftarrow kp_i + W_i(kp_i,\ k+1) + \sum_{\tau_j \in T_l} X_j \hat{e}_j$$

where $X_j = u_{-1}(\min(\lfloor \frac{(k+1)p_i}{p_j} \rfloor + 1, \lfloor \frac{R'_{i,k}}{p_j} \rfloor) - \lfloor \frac{kp_i}{p_j} \rfloor)$.

The iteration stops when $R_{i,k} = R'_{i,k}$ or $R_{i,k} \ge t + p_t$. Overall, the runtime complexity of the above two iterative procedures is $O(\lceil p_t/p_{min} \rceil)$ where $p_{min} = \min_{\tau_j \in T_l} p_j$. Since the size of S_i is proportional to $\lceil p_t/p_i \rceil$ and $\lceil p_t/p_i \rceil \le \lceil p_t/p_{min} \rceil$, the worst-case runtime complexity to construct a feature vector on $core_l$ at the PTM invocation is $O(|T_l| \cdot \lceil p_t/p_{min} \rceil^2)$.

B. Initialization of Model Parameters (Off-line Step)

The accuracy of SVM depends on the several unknown parameters used in the SVM. Hence, the model-selection procedure is required to determine such unknown parameters and to avoid the performance loss caused by over-fitting. For this purpose, we need a sufficient start-up period for initializing the model parameters before the system enters its normal operation. During this period, the PTM constructs a training data set which is only used for determining the model parameters in SVM. In this paper, the Gaussian kernel is chosen as the SVM kernel function, and 5-fold cross-validation is applied to determine the model parameters. The details of this procedure can be found in [10].

C. Model Training

At runtime, the thermal dynamics of a multi-core chip is affected by the chip's surrounding temperature. Since the room temperature can change during the system's mission, An online SVM is adopted for the PTM [11].

As described in Section II-D, the local scheduler updates the actual label of the feature vector over the last p_t time interval at the beginning of the PTM invocation. When no DTM is triggered over the last p_t interval, the PTM uses this information as a new training data and stores it in a queue to update the core's thermal model. For this, we assume that a thermal-model updating task τ_{ptm} runs on each core every p_t with the execution budget e_{ptm}. Like a polling server, it is scheduled with other tasks assigned on the core and consumes its execution budget every p_t to update the thermal model with a batch of new training data inputs stored in the queue. The thermal-model updating task suspends its execution when it consumes e_{ptm} time units or is suspended by other higher-priority tasks assigned on the core.

The thermal-model updating task executes two procedures, called the *incremental* and the *decremental learning*. When a new training data is added to the SVM, the incremental procedure updates the SVM from the previously-updated model. The decremental part is a pruning procedure which removes an old training sample from the existing training data set. When the ambient temperature changes, the thermal model should be able to adapt itself to the new environment by eliminating the out-of-date data from the training set. Also, this pruning part is essential to improve the runtime efficiency. Thus, the thermal-model updating task checks whether the number of training data is larger than N_{train}, which is given as a design parameter, after executing the incremental part. Then, it removes the least important support vector from the existing support vectors, and updates the SVM again.

V. Temperature Management Applicable to PTM

Using the temperature models, the PTM can detect whether the peak temperature will occur in a core proactively and determine when to cool it down at runtime. In this section, we briefly describe how the existing DTMs are incorporated with the PTM and applied to the cores that are predicted to be overheated.

A. DVFS

Chip-wide DVFS and per-core DVFS are the two main types of DVFS [12]. The chip-wide DVFS mechanism treats the entire chip as a single unit, while the per-core DVFS implements an on-chip voltage regulator on each core to isolate the power domain of a core from others'. To incorporate existing DVFS schemes in the PTM, we first introduce the concept of *job density* of a task τ_k, which is introduced in [13], and defined as

$$den_k(\mathbf{x}) = \begin{cases} \frac{c_k(\mathbf{x})\cdot f_{max}}{p_k} & \text{if the job released at } \mathbf{t} = \lfloor \frac{\mathbf{x}}{p_k} \rfloor p_k \\ & \quad \text{is completed before } \mathbf{t} = \mathbf{x}, \\ \frac{e_k \cdot f_{max}}{p_k} & \text{otherwise,} \end{cases}$$

where $c_k(\mathbf{x})$ is the actual execution time of a job of τ_k during $[\lfloor \frac{\mathbf{x}}{p_k} \rfloor \cdot p_k,\ (\lfloor \frac{\mathbf{x}}{p_k} \rfloor + 1) \cdot p_k]$, and it is obtained from the local scheduler. Since the clock speed should always be equal to or higher than the total sum of job densities of the tasks assigned on the core at any time [13], the unused cycles of the worst-case execution time are used to reduce the clock speed. Since the frequent voltage/frequency scaling results in switching overhead from the regulators, once the operating voltage/frequency of each core is adjusted at the current invocation time of the PTM, it remains unchanged until the next invocation of the PTM.

First, when per-core DVFS is available in the system, the sum of job densities of the tasks on $core_l$ at the PTM invocation time $\mathbf{t} = t$ is equal to $\sum_{\tau_k \in T_l} den_k(t)$. During $[t, t + p_t)$, the sum of job densities of the tasks changes at each job release time. By defining the job release list of $core_l$, denoted by $R_l(t,\ \mathbf{x})$, which contains a set of job release times during $[t,\ \mathbf{x}]$ on $core_l$, the lowest clock frequency, f_l, at which $core_l$ can operate is obtained as:

$$\max_{x \in R_l(t,\ p_t)} \sum_{\tau_k \in core_l} den_k(x) \leq f_l.$$

Thus, when $core_l$ is predicted to be overheated at time $\mathbf{t} = t$, the PTM adjusts the clock speed of $core_l$ during $[t, t + p_t)$ to the lowest clock frequency among all operating clock frequencies which are higher than f_l, and is set back to f_{max} at the next invocation of the PTM. Likewise, when chip-wide DVFS is applied, the clock speeds of all cores in $\mathbf{S}$ are adjusted to $\max_{core_l \in \mathbf{S}} f_l$.

B. Power-Gating

Power-gating is a well-known technique for efficient reduction of leakage power by inserting sleep transistors between logic transistors and power/ground [14]. Similarly, it can be implemented at either core level or chip level.

To determine how long the overheated core will be turned off, the PTM obtains the available time slack on the core at time $\mathbf{t} = t$, which can be calculated by using the similar methods in [15], [16]. If a core is predicted to be overheated, then the PTM turns it off during its available time slack. If it has some tasks to be released while the core is turned off, their releases are put off until the core wakes up after the cool-off. Like chip-wide DVFS, when the power domains of all cores are shared, and the entire chip needs to be turned off, the maximum allowable cooling time is determined by the core which has the minimum time slack.

VI. Performance Evaluation

The accuracy and the runtime overhead of the SVM-based thermal prediction model are evaluated while considering various model parameters and the runtime dynamics. Based on the thermal model, the performance of PTM with DVFS and power-gating is evaluated in terms of the percentage of peak temperature occurrence during runtime. This evaluation is done using the HotSpot thermal simulator with a set of synthetic tasks running on a target chip specified in the selected simulator.

A. Experiment Setup

We use the well-known HotSpot thermal simulator [7]. An 8-core chip is chosen as a target platform in which each core is set to operate at 1.5Ghz for its clock and 1.2V for its Vdd. The chip's layout is shown in Fig. 1, which replicates the Alpha 21264 floorplan with a scaled-down physical size of $12.6853mm^2$, considering 45nm CMOS process technology.

The application in the simulations consists of a set of synthetic tasks, each of which is generated with its period uniformly distributed in [20, 200]ms, and its utilization uniformly distributed in (0, 0.8]. The tasks are generated one-by-one and allocated to all cores using the First-Fit (FF) until the sum of task utilization in each core reaches 0.85, i.e., $\sum_{\tau_i \in T_l} e_i/p_i > 0.85$ for $\forall core_l \in \mathbf{S}$.

To model the realistic power dissipation on the target core model, we ran several embedded benchmark programs with various inputs in MiBench benchmark suites [17] on the Wattch power simulator [18], and obtained dynamic power dissipation samples during the execution of each application. Since the Wattch simulator does not support the selected target microprocessor and technology, we applied linear scaling methods (similar to those in [19]) to the Wattch-produced power traces for accurate estimation of dynamic power dissipation on the target core model. Then, the dynamic power characteristics of each application in MiBench benchmark are assigned randomly to the task in the task set. To obtain the leakage power consumption on the target core model, the McPAT power simulator [20] is used in our experiments.

Considering the fact that the convection resistance of the heat sinks for desktop processors is typically around 0.5~2K/W [21], we chose 1.0K/W as the convection resistance of the chip's heat sink. The temperature threshold Θ_{thres} is set to $85°C$. We assume that the room temperature is $45.15°C$ when the thermal profiles of the tasks and the unknown parameters in the SVM models are estimated before runtime.

B. Accuracy of Temperature Prediction Model

The accuracy of the prediction model is evaluated in terms of the missed detection rate (MDR) of overheating and the false alarm rate (FAR) for various runtime and design parameters. The MDR is derived by calculating the conditional probability, $P(Y^{\complement}|H)$, where H is the set of events at which the peak temperature actually occurs in a core during the PTM period and Y is the set of events when the core is predicted (by the model) to be overheated at that interval. Likewise, the false alarm rate is calculated as $P(Y|H^{\complement})$.

The design parameter, p_s, was set to 2ms. The prediction accuracy depends on the size of the training data set and the PTM period. Thus, the thermal prediction model was evaluated with $p_t = 50$ and 80ms, and different sizes of training data. In this paper, we only include the results with $N_{train} = 200$ and 300, because only a small improvement is shown to be made on the prediction accuracy when the size of training data grows larger than 300. In addition to the design parameters, we consider the following runtime conditions which are unknown, but affect the thermal behavior of a chip significantly at runtime.

- Variation of actual task execution time at runtime: The prediction accuracy is affected by the variation of the actual execution time of tasks. To account for this, we first measure the prediction accuracy with the strong assumption that the PTM has the exact knowledge of the actual execution time of all released jobs on cores and that the room temperature at runtime is fixed and identical to the temperature at design time without measurement errors. This is denoted as "ideal." However, the actual tasks' execution times are random variables, and it is very difficult to obtain their true distribution. So, we assume that the tasks' actual execution times follow a Gaussian distribution as in [22]. Specifically, we assume the ratios of tasks' actual execution times to worst-case execution times are normally distributed with mean 0.6 and standard deviation 0.2, which is denoted by C1.
- Different room temperatures at design and run times: The thermal profiles of tasks are determined at design time. Thus, the prediction accuracy is affected by the difference of the room temperature at design time and at runtime. For this, we set the room temperature to $50.15°C$, $5°C$ higher than the room temperature at design time, and increased the room temperature by $1°C$ every 20 minutes during our simulation. With the variation of the jobs' actual execution times, this condition is denoted by C2.
- Measurement noise: There may be measurement errors, thus making the prediction model learn inaccurately. The errors are mainly due to the inaccuracy and wrong placement of temperature sensors. To account for the impact of measurement errors, we added a random Gaussian noise with standard deviation $\sigma = 0.1$ or $0.5°C$ to the

temperature measurements. Each value is set to be the average error of the temperature measurement for sparse and dense architectures when 36 sensors are employed per core, respectively [6]. Overall, C3 denotes the case when all of the three conditions are considered together in the simulation.

For performance comparison, we consider the following methods.

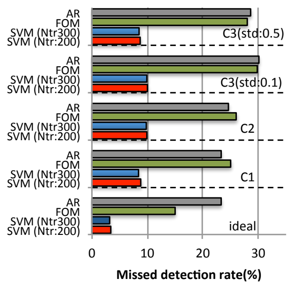
- <u>First-order model</u>: With the cores' temperature and the thermal profiles of the executed tasks recorded over time, the thermal model in Eq. 1 is obtained using a regression method. To estimate the model, only cores' temperatures are used, but the temperatures on other components on a chip are excluded from the regression process, because their temperatures cannot be measured. This method is denoted by FOM.
- <u>Linear AR method:</u> With this method, multiple linear AR models are devised on each core to predict the temperature during the next PTM period. The methods for the prediction model creation in [4] are adopted. We denote this method by AR.

For all parameters, the system continued operation for 120 simulation minutes after the core temperature got stabilized. The prediction model errors are plotted in Fig. 4.

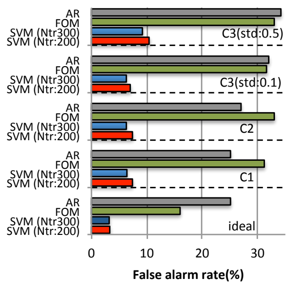
Overall, the MDR and FAR using the SVM-based predictor with $p_t = 50$ms in C1 is shown to be around 8.4–8.8% and 6.4–7.4%, respectively. Due to the randomness of the actual execution times of the jobs, the MDR and FAR of the SVM-based predictor is degraded up to 5.3% and 4.2%, respectively. Also, from the results in C2, when the room temperature changes slowly, the MDR of the SVM-based predictor is observed to be decreased by 1.1% due to the delay of adapting the model to the changed surrounding temperature. Also, the MDR is observed to be not affected by the and measurement error. However, in the presence of measurement errors, the FAR is increased by around 3.4% in this case.

When $p_t = 80$ms, FAR using the SVM-based predictor is increased by up to 7.7% and 8.1%. When p_t is increased, the percentage of training data with "overheated" is increased more than the case with a smaller p_t in the training set. Also, because more jobs are executed over a longer interval than a shorter one, the chance of creating incorrect feature vectors is increased as well due to the variation of the jobs' actual execution times. The possibility of being biased against non-overheated cases is thus increased. Note that a high FAR increases the overhead of DTM (e.g., the switching overhead of DVFS and the overhead from data retention when overheated cores are turned off), but the runtime overhead of the PTM is decreased, Thus, the value of p_t should be chosen by considering the tradeoff between the overhead of DTM trigger and the overhead of PTM invocation.

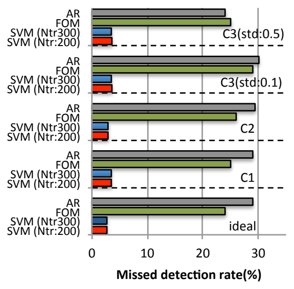
The results also show that the performance of AR and FOM. To achieve high accuracy with AR, the thermal behavior on the core should be a stationary process [4], which is not the case when multiple tasks with different power characteristics are executing with relatively short periods, which is usually on the order of few milliseconds. Also, because the temperature information of other internal thermal nodes on a chip cannot be measured, the simplified FOM cannot capture the thermal behavior of a multi-core chip correctly.



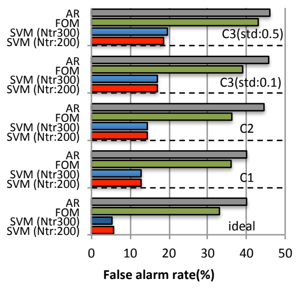
(a) Missed detection rate with $p_t =$ 50ms



(b) False alarm rate with $p_t = 50$ms



(c) Missed detection rate with $p_t =$ 80ms



(d) False alarm rate with $p_t = 80$ms

Figure 4. Missed detection and false alarm rates of the thermal model of an 8-core chip.

C. Runtime Overhead

Upon availability of new training data, the thermal model of a core is updated by the thermal-model update procedure on the core with period p_t and execution budget e_{ptm}. Thus, to determine the proper value of e_{ptm}, we should consider the runtime overhead of the incremental and decremental procedures in the SVM, which is affected by the number of training samples and the dimension of the feature vector. Considering these, we measured the runtime overhead of the training procedure in the SVM by executing the algorithm on the SimpleScalar micro-processor simulator [23]. The processor speed is set to 1.5Ghz, which is the same as the core speed used in our simulation, and the results are plotted in Fig. 5. When $N_{train} \leq 200$, the time required for updating the thermal model is less than 1.5–2.0 ms when $p_t = 50$ and 80 ms. Thus, by allocating 3.9% and 2.5% of the total utilization to the thermal-update procedure/task in each core, the PTM is able to update the thermal model when the surrounding temperature varies at runtime.

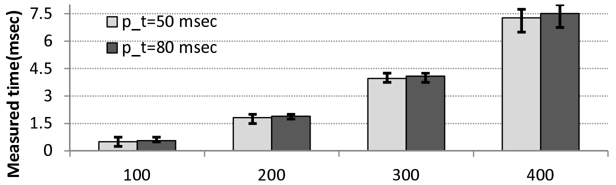


Figure 5. The runtime overhead of updating the thermal model with a new training data with $p_t = 50$ and 80 ms

D. Percentage of Temperatures Exceeding Threshold

We now evaluate the performance of DVFS and power-gating in the PTM considering all conditions in C3 with $\sigma = 0.5$ and $N_{train} = 200$. The metric used for performance comparison is the statistics of the sampled core temperatures exceeding the threshold during runtime. When we ran the given task set on the multi-core chip without DTM, the percentage of the total sampled core temperatures above the threshold was found to be around 14.7% on average for a sufficient long period. We assume that there are four DVFS levels: 0.8, 1.0, 1.2 and 1.5 GHz. Fig. 6 shows the percentage of the sampled core temperatures exceeding the threshold when different DTM schemes are applied to the cores.

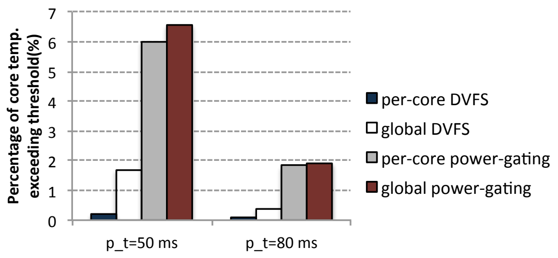


Figure 6. The percentage of sampled core temperatures.

When per-core DVFS is applied, the percentage of temperatures exceeding the threshold was observed only to be 0.048% with $p_t = 80$ms, which is much smaller than 0.195% with $p_t = 50$ms. However, with $p_t = 80$ms, the PTM unnecessarily adjusts the operating frequency of cores due to a high false-alarm rate.

Also, when the global DVFS is used in the PTM, the percentage of peak temperatures is increased by up to 1.7% and 0.4% when $p_t = 50$ or 80ms, respectively. The performance is degraded due to the existence of cores with minimum time slack when other cores are predicted to be overheated. Overall, the performance of power-gating is worse than the global DVFS, and the performance difference is around 4.3% or 1.4% when $p_t = 50$ or 80ms, respectively.

VII. Related Work

A large body of research has been done on power and temperature management in real-time and/or embedded systems. Chen *et al.* [2] proposed a proactive real-time scheduling algorithm on a uniprocessor that minimizes the job completion times under the maximal thermal constraints, or minimizes the temperature under both the timing and maximal thermal constraints. Based on the simple chip thermal model, approximated by Fourier's Law, a processor speed schedule was derived statically. Similarly, Fisher *et al.* [3] studied thermal-aware scheduling on a multi-core platform, in which the optimal speed schedule of cores is derived, assuming that the thermal model of a multi-core chip is given at design time. Using similar assumptions and models, Wang *et al.* [24] proposed another speed scheduling algorithm that minimizes the energy consumption subject to thermal and timing constraints.

Using these approaches, the both thermal and timing constraints are guaranteed to be met at system design time, only when the thermal model of a chip is correctly estimated and given at system design time. However, even though these thermal models are correctly estimated at system design time, the actual thermal behavior of a chip could differ significantly from those estimated at design time, because the room temperature may differ at runtime. Also, the power consumption of computationally-intensive tasks is higher than that of others with less intensive workloads. However, most of these papers do not consider such non-uniform power/thermal characteristics of tasks.

For temperature management at runtime, Fu *et al.* [25] proposed a control-theoretic algorithm to meet the desired temperature requirement on a multi-core chip subject to timing constraints. However, since it does not include temperature prediction, it cannot avoid the occurrence of peak temperature during the sampling period. Wang *et al.* [26] also applied control theory to power and temperature management without considering real-time constraints.

For temperature prediction models, Coskun *et al.* [4] studied the autoregressive moving average (ARMA) model to predict the future temperature of a processor, but they assume that the phases of the given workload are stationary processes, which does not hold for the systems where multiple tasks with different thermal characteristics are invoked periodically and scheduled together on the same core. Zhang *et al.* [27] proposed a new method for tracking statistical characteristics of the rocessor's power state and predicting any change of the power state using Kalman filter. However, none of these addresses the timing constraints of mission-critical applications.

VIII. Conclusions

In this paper, we presented a new, runtime peak temperature manager to meet both timing and thermal constraints of real-time applications running on a multi-core platform. The PTM uses a proactive mechanism to predict the future thermal dynamic behavior of a target multi-core chip using SVM-based prediction models. The models are based on the thermal profiles of the tasks obtained off-line, which are, in turn, used to determine when to apply the existing DTMs to overheated cores to meet the thermal constraints without violating the application timing constraints. Also, using the runtime information, the PTM trains the thermal models of a multi-core chip online, and predicts the future thermal behavior of each core at its invocation time.

We have evaluated the effectiveness of our PTM using a combination of several widely-used power/thermal simulators and a set of synthetic tasks with realistic power characteristics obtained from well-known testbench programs running on an 8-core chip model with 45nm technology. Our evaluation results have shown that the proposed models can predict the occurrence of peak temperature in a core with the miss rate of 8.8–10.0% and the false alarm rate of 6.3–9.2% at the expense of system resource utilization of 3.0–3.9%, which is used to update the thermal models of a multi-core chip. Given the encouraging results of our PTM, we plan to conduct an extensive evaluation of the proposed solution for more complex cases. Other hybrid DTM methods under various system assumptions will also be incorporated into the proposed PTM.

Acknowledgement

The work reported in this paper was supported in part by the NSF under Grant No. 0930813 and 113820.

References

[1] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The impact of technology scaling on lifetime reliability," in *IEEE Computer Society*, 2004.

[2] J.-J. Chen, S. Wang, and L. Thiele, "Proactive speed scheduling for real-time tasks under thermal constraints," in *Proc. of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, 2009.

[3] N. Fisher, J.-J. Chen, S. Wang, and L. Thiele, "Thermal-aware global real-time scheduling on multicore systems," in *Proc. of the 2009 15th IEEE Symposium on Real-Time and Embedded Technology and Applications*, 2009.

[4] A. K. Coskun, T. S. Rosing, and K. C. Gross, "Proactive temperature management in mpsocs," in *Proc. of the 13th international symposium on Low power electronics and design*, 2008.

[5] I. Yeo, C. C. Liu, and E. J. Kim, "Predictive dynamic thermal management for multicore systems," in *Proc. of the 45th annual Design Automation Conference*. ACM, 2008.

[6] J. Long, S. O. Memik, G. Memik, and R. Mukherjee, "Thermal monitoring mechanisms for chip multiprocessors," *ACM Trans. Archit. Code Optim.*, vol. 5, 2008.

[7] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. Stan, "Hotspot: a compact thermal modeling methodology for early-stage vlsi design," *VLSI Systems, IEEE Trans. on*, vol. 14, no. 5, 2006.

[8] Y. Han, I. Koren, and C. M. Krishna, "Tilts: A fast architectural-level transient thermal simulation method." *J. Low Power Electronics*, vol. 3, no. 1, pp. 13–21, 2007.

[9] E. Rotem, A. Naveh, M. Moffie, and A. Mendelson, "Analysis of thermal monitor features of the intel pentium m processor," in *in Workshop on Temperatureaware Computer Systems*, 2004.

[10] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A Practical Guide to Support Vector Classification," 2000.

[11] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," 2000.

[12] W. Kim, M. S. Gupta, G. yeon Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *International Symposium on High-Performance Computer Architecture*, 2008.

[13] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced cpu energy," in *Proc. of the 36th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1995.

[14] B. Calhoun, F. Honore, and A. Chandrakasan, "A leakage reduction methodology for distributed mtcmos," *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 5, 2004.

[15] H. Chetto and M. Chetto, "Some results of the earliest deadline scheduling algorithm," *IEEE Trans. Softw. Eng.*, vol. 15, no. 10, pp. 1261–1269, 1989.

[16] T.-S. Tia, "Utilizing slack time for aperiodic and sporadic requests scheduling in real-time systems," Ph.D. dissertation, 1996.

[17] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proc. of the Workload Characterization, 2001 IEEE International Workshop*. IEEE Computer Society, 2001.

[18] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proc. of the 27th Annual International Symposium on Computer Architecture*, 2000.

[19] R. Kumar, K. Farkas, N. Jouppi, P. Ranganathan, and D. Tullsen, "Single-isa heterogeneous multi-core architectures: the potential for processor power reduction," in *Proc. of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 81 – 92.

[20] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Micro. 42nd Annual IEEE/ACM International Symposium on*, 2009.

[21] R. Viswanath, V. Wakharkar, A. Watwe, V. Lebonheur, M. Group, and I. Corp, "Thermal performance challenges from silicon to systems," 2000.

[22] J. R. Lorch and A. J. Smith, "Improving dynamic voltage scaling algorithms with pace," in *Proc. of the 2001 ACM SIGMETRICS international conf. on Measurement and modeling of computer systems*. ACM, 2001.

[23] T. Austin, E. Larson, and D. Ernst, "Simplescalar: an infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, 2002.

[24] S. Wang, J.-J. Chen, Z. Shi, and L. Thiele, "Energy-efficient speed scheduling for real-time tasks under thermal constraints," in *Proceedings of the 2009 15th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, ser. RTCSA '09, 2009, pp. 201–209.

[25] Y. Fu, N. Kottenstette, C. Lu, and X. D. Koutsoukos, "Feedback thermal control of real-time systems on multicore processors," in *Proc. of the tenth ACM international conference on Embedded software*, ser. EMSOFT '12. ACM, 2012.

[26] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *Proc. of the 36th annual international symposium on Computer architecture*, ser. ISCA '09. ACM.

[27] Y. Zhang and A. Srivastava, "Adaptive and autonomous thermal tracking for high performance computing systems," in *Proc. of the 47th Design Automation Conference*. ACM, 2010, pp. 68–73.