On-Line Predictive Thermal Management under Peak Temperature Constraints for Practical Multi-core Platforms

Guanglei Liu[1], Ming Fan[1], Gang Quan[1], Meikang Qiu[2]

[1]Department of ECE, Florida International University, Miami, FL 33174

[2]Department of ECE, University of Kentucky, Lexington, KY 40506

gliu002@fiu.edu, mfan001@fiu.edu, gang.quan@fiu.edu, mqiu@engr.uky.edu

*corresponding author: Guanglei Liu

Address:

Florida International University

Department of Electrical and Computer Engineering

10555 West Flagler Street, EC 3155

Miami, FL 33174, USA

Office : (305) 348 3715

Fax : (305) 348 3707

Email : gliu002@fiu.edu

Date of Receiving: **to be completed by the Editor**

Date of Acceptance: **to be completed by the Editor**

On-Line Predictive Thermal Management under Peak Temperature Constraints for Practical Multi-core Platforms

Guanglei Liu, Ming Fan, Gang Quan, Meikang Qiu

Abstract - *The power and thermal issues have become one of the major design challenges for the development of modern computing systems. Developing an effective Dynamic thermal management algorithm for the practical multi-core computing system to deliver maximal throughput without encountering temperature emergencies becomes highly demanded. In this paper, we first identify the limitation of the existing theoretical work, and we introduce an enhanced reactive thermal management algorithm based on the* dynamic voltage and frequency scaling *(DVFS) technique. Then, we develop a new temperature prediction technique and migration scheme that take the local temperature of a core as well as the impacts from neighboring cores into consideration and we validate our algorithm on an Intel desktop. The experimental results show that our approach can significantly improve the throughput while satisfying the temperature constraint compared to the conventional approach.*

Keywords - Dynamic thermal management, thermal-aware scheduling, throughput maximization, temperature prediction, thermal-aware management.

1 INTRODUCTION

Due to the increasing demand for the higher computation capability, the size of transistors is continuously shrinking, and more and more transistors are integrated into a single chip to build up more complicated circuit architectures, i.e. *chip multiprocessors* (CMPs). As a result, the power density of the IC chip exponentially increases, and also generates a large amount of heat. The rapidly growing heat generation greatly increases the packaging/cooling costs, and adversely affects the performance and reliability of a computing system. Besides, the increased heat generation may reduce processor life span, even force the computing system to completely shut down to prevent permanent physical damage to the processor [25]. Therefore, an effective thermal management solution is highly desirable, not only to balance the chip's temperature but also to enable the computing system to operate at a high computing performance without exceeding its temperature limit.

Dynamic thermal management (DTM) technique, as one of the most effective approaches to address the power and thermal design issue, has been researched extensively in recent years. Some of the researches are focusing on the thermal aware performance maximization problem [5, 31, 32, 18, 21]. To solve this problem, the DVFS technique has been widely used to develop the thermal-aware DTM algorithms [16, 15, 24, 11, 20]. It can control the temperature by dynamically adjusting the processor speed based on the workload. Thus, some researches have been done based on this characteristic. Zhang and Chatha [31] presented a pseudo-polynomial time speed assigning algorithm based on dynamic programming to minimize the total execution latency. They further developed several heuristics [32] to maximize the throughput of a real-time system by sequencing the execution of a task set consisting of tasks with different power and thermal characteristics for processors with and without DVFS capabilities. Chantem et al. [5] proposed one methodology to run real-time tasks by frequently switching between the two speeds which are neighboring to the constant speed whose stable temperature is the given peak temperature limit. However, DVFS techniques sacrifice the performance to cool down the temperature. Task migration is an alternative technique to manage the temperature by balancing the workload among CPU cores without slowing down the processing speed [30, 15, 12, 29, 9, 23, 19]. For example, Fabrizio et al. [22] presented a lightweight thermal balancing policy, which could minimize the on-chip temperature gradients by using the task migration technique. Coskun [10] et al. proposed a thermal aware scheduling algorithm, which migrates tasks according to the calculated priority. However, the above theoretical works are based on simplified models and idealized assumptions, such as the accurate temperatures of processors are readily available, which is not necessarily true on a practical platform.

When DTM techniques are applied for real applications, they must deal with important practical details in the physical environment. To this end, many researches have been carried out based on practical hardware platforms [3, 6, 4, 21, 28, 14]. For example, Yefu [27] et al. proposed a chip-level power management algorithm by using control theory and implemented their algorithm on an Intel Xeon desktop. Ahn [3] et al. developed and validated a heuristic to reduce the power consumption and control the temperature on the Intel Centrino Duo and ARM-11 MPCore platforms. The above algorithms rely on the thermal sensor reading to trigger their DTM actions. Since the thermal sensor lacks accuracy due to their placement location and long latency, the effectiveness of the DTM techniques can be severely

degraded. To address this issue, performance counters have been used as a soft sensor to develop more accurate thermal modules [7, 17]. Even though the thermal sensor can accurately detect a thermal emergency when temperature reaches the threshold, it still takes 100 to 200 millisecond for the DTM manager to decrease the frequency or migrate the hot task to a different processor [16]. As a result, the temperature would exceed the threshold before the algorithm takes effect. To this end, predicting the potential thermal emergency before thermal failure occurring is a very important feature for the DTM algorithm [12]. In response to this, Inchoon [30] et al. proposed a temperature prediction algorithm, which takes the application's thermal behavior into consideration. Khan [15] et al. developed an alternative thermal management schedule which combined temperature history based prediction and task migration techniques to efficiently control the CPU temperature under threshold. However, they assumed that at each sampling point, the temperature will increase at the same rate until it reaches the threshold, which is not true for the practical scenario.

In this paper, we first identify several limitations of existing theoretical algorithms on the practical desktop platform and develop a reactive thermal aware DVFS algorithm to evaluate the efficiency of *dynamic voltage and frequency scaling* technique on the practical computing system. Then, we analyze the drawbacks of the reactive DTM approach and propose an on-line predictive thermal management algorithm to maximize the throughput on multi-core systems while satisfying the peak temperature constraint. Compared with the previous work, we make three major contributions in this work:

- We develop a temperature prediction method, which can predict the temperature of a core more accurately by taking its temperature as well as the neighboring impacts into consideration.
- We develop a new task migration strategy. While it has been a common approach to migrate tasks from the hottest to the coolest core, our approach chooses the destination core differently. We choose the destination core not only by its current temperature, but also by the temperature trends as well as the neighboring impacts as well.
- We validate our algorithm on a practical hardware test bed, i.e. a desktop workstation with an Intel i5 750 quad core microprocessor. The experimental results show that our proposed algorithm can significantly outperform the previous approachs.

The rest of the paper is organized as follows. In Section 2, we first discuss the limitation of the theoretical work, and then use an example to motivate our research. To overcome the hardware limitations, we introduce an enhanced reactive approach in Section 3. We present the neighbor-aware temperature prediction technique in Section 4, and propose our predictive thermal-aware algorithm in Section 5. Experimental results are discussed in Section 6, and the conclusion is given in Section 7.

2 PRELIMINARY

In this section, we first introduce the related theoretical work on throughput maximization and their limitations. Then we give motivational examples to introduce the design constraints to develop an effective dynamic thermal management algorithm.

2.1 Motivational example

As discussed in Section 1, many researches have been published to address the thermal aware performance maximization problem [5, 31, 32]. However, most of these approaches require detailed knowledge of processes running on the platform, such as the exact numbers of processes and their execution times, which is not always available on a general computing platform such as the desktop. Some other information such as the thermal resistance and the thermal capacitance, which are essential to build the thermal model for the processor, are also not immediately available.

To develop a general thermal aware scheduling algorithm for the desktop computing platform, the less information is required regarding the processes and platform, the more effective can the algorithm be. In this regard, the *reactive two speed* scheduling approach, proposed by Wang et al [26], seems to be a good choice. According to this algorithm, the processor runs at the maximum speed before it reaches the temperature limit and then runs at a speed, so called *the equilibrium speed*, to maintain this temperature. It has been proved [8], theoretically, this is the optimal approach to maximize the workload under a peak temperature constraint.

However, there are several drawbacks in this approach. First, since most processors support only discrete levels of working frequencies. The ideal equilibrium speed may not always be available for a given peak temperature constraint. Second, the power consumption of the processor varies with not only the processor speeds, but also other factors such as the types of processes, operating temperature, etc. In fact, even a single processor running a single process may have different power consumptions at different times [16]. Therefore, the *the equilibrium speed* is not unique and constant at all, and it is simply not possible to simply set a processor to a constant speed once and for all to maintain a constant temperature.

Dealing with these problems, another approach [25], as shown in Figure 1, seems to be more flexible. This approach assumes no a priori knowledge of the applications running on the computing system at all. It monitors the chip temperature regularly and adjusts the processor speed dynamically. At each temperature sampling point, if the current temperature does not reach the threshold, the processor speed is elevated to one level higher. Otherwise, if the current temperature equals or exceeds the temperature limit, the processor speed is changed to one level lower to cool down the temperature.

At first sight, it seems that this approach solves all the problems mentioned above. It naturally assumes the processor has a discrete working frequency levels. It does not assume any a priori knowledge of the programs running on the processor either. However, there are still a few problems that make this approach less effective in a practical desktop environment. First, this approach assumes that the instant temperature information is available immediately and accurately. Second, updating the frequency level one at a time might not be quick enough to respond to temperature change and meet the temperature constraint.

We use a simple example to explain these two problems. Consider Figure 2. Recall that it takes about 1 second for the thermal sensor in our desktop to reflect any temperature changes. It is possible that even though the system temperature has already reached or surpassed the temperature threshold at t_1, the sensor reading may still be lower than temperature limit. The algorithm continues to increase the speed level of the processor and thus violates the

peak temperature constraint. Moreover, even though the algorithm can sense the accurate temperature at t_1 and find that it has already reached the temperature threshold, since it adjusts frequency level one at a time, it may not be able to reduce the temperature fast enough. The temperature continues to rise and violates the peak temperature constraint. To address these problems, in what follows, we develop a reactive and a proactive DTM algorithm respectively.

2.2 Problem description

The system considered in this paper consists of N tasks, denoted as $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$ and M identical processors, denoted as $\mathcal{P} = \{P_1, P_2, ..., P_M\}$. The problem discussed in this paper is how to manipulate the scheduler such that the throughput of the system can be maximized under peak temperature constraint. The formal description of the problem is represented below.

Given a task set Γ and a multi-core system $\mathcal{P}$, maximize the throughput of the system under the peak temperature constraint, denoted as $T^{THRESHOLD}$.

For processor P_i, we use a tuple (T_i, t_i) to represent the temperature of P_i at a certain time point t_i. To be more specific, we use T_i^{curr} and T_i^{prev} to denote P_i's current temperature and previous temperature respectively, while t_i^{curr} and t_i^{prev} are the corresponding times.

In order to address the design constraints described in the previous subsection, we first introduce an improved reactive DTM heuristic. We implement a non-constant sampling period based on the practical hardware platform characteristic to reduce the response latency. Also, by defining the temperature buffer zone and building up an offline temperature speed lookup table, the algorithm quickly adjust to a proper working frequency level to improve the throughput.

After we discuses the limitation of the reactive approach, we propose our predictive thermal management algorithm. We first introduce a new temperature prediction method, which predicts the future temperature of a processor core by considering both local temperature history and neighbors' effect. Once a potential risk is detected under our temperature prediction model, i.e. the predicted temperature is over the threshold, we dynamically manage the executions of corresponding tasks on that core by either migration or DVFS. By considering the neighbors's temperature and their temperature changing trends, we can select a processor among all available candidates to improve the total system performance from a global and long-term perspective.

3 ENHANCED REACTIVE DYNAMIC THERMAL MANAGEMENT ALGORITHM

In this section, we give a detailed introduction about our Enhanced Reactive Dynamic Thermal Management (ERDTM) algorithm, which can maximize the system throughput while satisfy the temperature constraint.

3.1 Identify sampling period

Being able to monitor the temperature change timely and accurately so that the thermal management algorithm can react is one of the most critical issues for the reactive approach. Thus, defining the appropriate sampling period becomes critical. One intuitive idea to define

the sampling period is to set the period as small as possible in order to track the temperature change quickly. However, the redundant sensor reading can cause accumulated overhead and degrade the overall system performance. Meanwhile, it takes extra power to read thermal sensor and increase the chip temperature. On the other hand, setting the sampling period too long cannot catch the temperature variations timely.

Given the limitations of the temperature sensor in our platform, we set the sampling period equal to the minimal response time of the thermal sensor for temperature change. To identify the minimal temperature response time, we ran different benchmarks at different speeds with different sampling periods. The minimal interval within which the temperature sensor has the same reading is set to be the sampling period. From our empirical work, we found the minimal temperature response time is 0.98 seconds. However, in the worst-case scenario, it cannot always take 0.98 seconds to find out that the thermal sensor readings has changed. For example, the thermal reading changes exactly after one sampling point. To further improve the performance, we use non-constant sampling periods. It uses two sampling speeds, *regular sampling speed*, denoted as $\mathcal{P}_{regu}$ and *small sampling speed*, denoted as $\mathcal{P}_{smal}$, where $\mathcal{P}_{regu} >> \mathcal{P}_{smal}$. When temperature changes are detected at the sampling point, a new processor speed will be set accordingly and the sampling period remains as $\mathcal{P}_{regu}$. If the temperature sensor remains the same value, (i.e. $T^{curr} = T^{prev}$), the algorithm changes the sampling period to P_{smal} and processor speed remain the same. In this work, $\mathcal{P}_{regu} = 0.98$ seconds and $\mathcal{P}_{smal} = 0.1$ seconds accordingly. In comparison with the algorithm using a constant sampling period, this approach catches temperature change and responds to it more timely.

3.2 Buffer zone and lookup table

Even though the non-constant sampling period can effectively detect a temperature change, it still takes time for the conventional reactive approach to adjust the processor speed one level at a time. Thus, when the temperature is really close to its limit, the processor speed is not decreased fast enough to cool down the temperature in time. On the other hand, when the temperature is much lower than the temperature threshold, the processor speed is not increased fast enough to maximize the throughput.

To solve these problems, we first introduce a concept called the *temperature buffer zone* as shown in Figure 3. Given a temperature threshold $T^{THRESHOLD}$, the temperature buffer zone is defined as the interval of $[T^{SAFE}, T^{THRESHOLD}]$, where T^{SAFE} is determined by the following equation

$$T^{SAFE} = T^{THRESHOLD} - \triangle T, \tag{1}$$

where $\triangle T$ is the maximum possible temperature increment within one sampling period. $\triangle T$ can be determined empirically. Using SPEC2000 benchmark, we found that $\triangle T = 4^oC$. When the temperature is lower than T^{SAFE}, we say that the temperature is in the *safe region*.

When the processor temperature is within the safe region, we can safely use the highest possible speed to maximize the throughput before temperature enters into the buffer zone. Thus, the problem becomes how to define the safe speed to run the task and ensure the temperature does not exceed the threshold after entering the buffer zone.

To solve this problem, we implement an offline thermal profiling analysis by collecting

their temperature information associate with different execution speed. Thus, given a task set $\Gamma = \{\tau_1, \tau_2, ..., \tau_N\}$, $T_{stable}(\tau_i, s_j)$ denotes the stable temperature when running τ_i using processor speed s_j. Let S_i be the speed such that

$$S_i = \max\{s_j \text{ such that } T_{stable}(\tau_i, s_j). \leq T^{THRESHOLD}\}. \tag{2}$$

And the safe speed S_{safe} is determined as follows.

$$S_{safe} = \min_{\tau_i \in \mathcal{T}} S_i. \tag{3}$$

Our Enhanced Reactive Dynamic Thermal Management algorithm is depicted in Algorithm 1. When a process is running, the processor uses the highest possible speed to improve its throughput if the temperature is located in the *safe region* (line 5-6). Otherwise, it adopts the safe speed to make sure the temperature constraint is not violated (line 7-9). If the running task sets are known a priori, we can further improve the performance of our algorithm by building up a lookup table. The lookup table lists the tasks and their specific safe speeds under different temperature constraints, as those defined in equation (2). In that case, we can use the corresponding safe speed depending on the current running process to further maximize its throughput.

Algorithm 1 ENHANCED REACTIVE DYNAMIC THERMAL MANAGEMENT

```
1: while Process is running do
2:   if T^curr = T^prev; then
3:     Wait P_smal = 0.1 seconds;
4:   else
5:     if T^curr ≤ T^SAFE then
6:       Set processor speed to the maximum speed;
7:     else
8:       Set processor speed to S_safe.
9:     end if
10:    Wait P_regu = 0.98 seconds;
11:  end if
12: end while
```

4 NEIGHBOR-AWARE TEMPERATURE PREDICTION

As we discussed in Section 3, the reactive approach might not precisely react with the temperature change. Thus, an effective temperature prediction heuristic, which can accurately detect the temperature emergency, is highly demanded. In this section, we first use a motivation example to illustrate the importance of considering heat transfer from neighboring processor when developing temperature prediction algorithm, then we proposed two different neighbor-aware temperature prediction techniques.

4.1 Motivational example

The processor heat dissipation comes mainly from the power consumed by the processor. However, there is another important heat source that comes from the neighboring cores which cannot be ignored. Since the number of transistors and cores that are integrated into the CMPs chip is increasing, the power density rapidly increases. Each core also receives heat transferred from its surrounding neighbor. This heat can also heat up a core, even though it is not running at the highest working frequency.

To illustrate this scenario, we executed one set of experiments to study how different neighbor environments can affect the processor temperature. First, we selected one core of our multi-core platform, for which its working frequency was set to the minimal speed level without executing any benchmark. Then, the temperature traces of this *idle* core with two different neighbor environments are collected. With the *Hot neighbor* environment, the surrounding cores have been assigned with the highest working frequency and executing a *hot* process to create a high temperature neighbor environment. On the other hand, with *Cool neighbor* environment, all the neighbor cores have been assigned with low working frequency running a *cool* task. The temperature information of the idle core with two different neighbor environments is collected and plotted in Figure 4. The experimental result clearly shows that even when the idle core does not execute any process, the heat transfer from the neighboring cores can also heat up its temperature by as much as 18^oC (i.e. 61^oC at the stable state) with the hot neighbor environment. In contrast, the idle core temperature only increased 5^oC (i.e. 45^oC at the stable state) with the cool neighbor environment.

4.2 Temperature prediction model

In this subsection, we introduce the temperature prediction model, which takes the heat transfer from the neighboring processors into consideration. It can accurately predict the future temperature of a core as well as its future trend. First, we introduce the following definitions to represent the future local temperature increment of each processor individually.

Definition 4.1. *Given processor P_i, the* local increment *factor of P_i, denoted as $\mathcal{I}_i^{in}$, is defined as*

$$\mathcal{I}_i^{in} = T_i^{curr} - T_i^{prev}. \tag{4}$$

This local temperature increment will be used to predict the future temperature at the next sampling point, as shown in Figure 3 (Δt is the sampling period). In this work, the sampling period has been set to 1 second, since this is approximately how long it takes for the thermal sensor to reflect a temperature change [2].

Besides its own power consumption, the temperature of a processor is also affected by other processors on the same chip, especially its neighbors. In this paper, we define the *neighbor processors* of a processor P_i, denoted as $\mathcal{P}_i^{NB}$, as the cores which are adjacent to P_i. When predicting the temperature of a processor, we consider only the heat transfer impacts from its neighboring processors to simplify our algorithm. By considering the effect of neighbor processors, we define the following two concepts to represent the neighbors' thermal effect for a given processor. The first concept, i.e. *neighbor average* factor, represents the average temperature of all neighbors. The second concept, i.e. *neighbor increment* factor, represents the temperature increment trend of all neighbors. Two concepts are formally defined as follows.

Definition 4.2. *Given any processor P_i, the* neighbor average *factor of P_i, denoted as $\mathcal{A}(P_i)$, is defined as*

$$\mathcal{A}(P_i) = \frac{\sum_{P_j \in \mathcal{P}_i^{NB}} T_j^{curr}}{|\mathcal{P}_i^{NB}|} \tag{5}$$

Note that $|\mathcal{P}_i^{NB}|$ returns the number of neighboring processors of P_i.

Definition 4.3. *Given any processor P_i, the* neighbor increment *factor of P_i, denoted as $\mathcal{I}(P_i)$, is defined as*

$$\mathcal{I}(P_i) = \frac{\sum_{P_j \in \mathcal{P}_i^{NB}} (T_j^{curr} - T_j^{prev})}{|\mathcal{P}_i^{NB}|} \tag{6}$$

According to Definition 4.3, $\mathcal{I}(P_i)$ represents the average temperature increment of P_i's neighboring processors. In other words, the neighbor increment factor describes the temperature increment speed for each processor's neighbors.

Consider processor P_i, the temperature increment caused by P_i's neighbors can be calculated as following

$$I_i^{nb} = \gamma_1 \cdot \mathcal{A}(P_i) + \gamma_2 \cdot \mathcal{I}(P_i), \tag{7}$$

where γ_1 and γ_2 are the weights of $\mathcal{A}(P_i)$ and $\mathcal{I}(P_i)$, respectively, which can be obtained off-line.

With the above definitions, we are now ready to introduce our temperature prediction model. Let T_i^{pred} denote the predicted temperature for P_i. We formulate T_i^{pred} as a linear function of its current temperature T_i^{curr}, its local temperature increment rate I_i^{in}, and also its neighbor effect factor I_i^{nb}, as shown below:

$$T_i^{pred} = \alpha_i \cdot T_i^{curr} + \beta_i \cdot \mathcal{I}_i^{in} + \gamma_i \cdot \mathcal{I}_i^{nb}, \tag{8}$$

where α_i, β_i and γ_i are weight parameters for P_i.

In addition, to make our prediction model more accurate, we take different processor location scenarios into consideration. Because each processor with different number of neighboring cores has different neighbor effects as shown in Figure 5. Thus, the temperature prediction for a task τ_i can be categorized into three cases: 1) τ_i runs on a corner processor; 2) τ_i runs on a boundary processor; 3) τ_i runs on a middle processor. Then we discuss the neighbor effect for τ_i by using matrix.

Temperature prediction base $\hat{T}_i^B$ is a 3×1 vector:

$$\hat{T}_i^B = [T_i^{curr},\ I_i^{in},\ I_i^{nb}]^T.$$

Based on the different cases of processor position, i.e. corner, boundary and middle, temperature prediction weights for different scenario can be expressed as following.

- weights for corner scenario:

$$w_i^c = [\alpha_i^c,\ \beta_i^c,\ \gamma_i^c].$$

- weights for boundary scenario:

$$w_i^b = [\alpha_i^b,\ \beta_i^b,\ \gamma_i^b].$$

- weights for middle scenario:

$$w_i^m = [\alpha_i^m,\ \beta_i^m,\ \gamma_i^m].$$

Combine all three scenarios of τ_i together, we have

$$W_{i\ 3\times3} = [w_i^c,\ w_i^b,\ w_i^m]^T.$$

Next, we introduce two temperature prediction algorithms based on the above prediction model.

4.3 Neighbor-different prediction

In this subsection, we introduce a *neighbor-different temperature prediction* (NDTP) algorithm, which considers all the different scenarios of neighbor processor condition as discussed in the previous subsection. We conduct the temperature prediction matrix, i.e. $\hat{T}_i$ to represent the temperature prediction result for task τ_i.

$$\hat{T}_{i\ 3\times1} = [T_i^c,\ T_i^b,\ T_i^m]^T,$$

where T_i^c, T_i^b and T_i^m are the temperature prediction results for corner processor, boundary processor and middle processor, respectively.

For each item of $\hat{T}_i$, i.e. T_i^x, $x \in [c, b, m]$, the temperature can be calculated by

$$T_i^x = [\alpha_i^x,\ \beta_i^x,\ \gamma_i^x] \times [T_i^{curr},\ \Delta T_i^{in},\ \Delta T_i^{nb}]^T = w_i^x \times B_i, \tag{9}$$

thus, we have

$$\begin{bmatrix} T_i^c \\ T_i^b \\ T_i^m \end{bmatrix} = \begin{bmatrix} \alpha_i^c & \beta_i^c & \gamma_i^c \\ \alpha_i^b & \beta_i^b & \gamma_i^b \\ \alpha_i^m & \beta_i^m & \gamma_i^m \end{bmatrix} \times \begin{bmatrix} T_i^{curr} \\ \Delta T_i^{in} \\ \Delta T_i^{nb} \end{bmatrix} \tag{10}$$

or

$$\hat{T}_i = W_i \times B_i. \tag{11}$$

Since we can get the weight matrix W_i off-line, the predicted temperature of τ_i can be obtained on-line by determining host processor position of τ_i.

The detail flow of *NDTP* algorithm is presented in Algorithm 2. For any processor P_i, we first get the current temperature from the sensor and previous temperature from history record, and denote as T_i^{curr} and T_i^{prev} respectively, as shown in line 1 and line 2. Then, based on the current and history temperatures, we estimate the local temperature increment for the next sampling period, i.e. Δt, under the same trend of nearest history (line 3). Moreover, we take the environment of P_i into consideration by calculating the neighbors' thermal effect of P_i (line 4). The weight factors can be determined by identifying the processor's location (line 6). Then we predict the future temperature according to the three major factors (line 7).

4.4 Neighbor-normalized prediction

Instead of categorizing the processors into three categories and generating three different groups of weight, we propose a *neighbor-normalized temperature prediction* (NNTP) algo-

Algorithm 2 Neighbor-Different Temperature Prediction

1: T_i^{curr} := the current temperature of processor P_i;
2: T_i^{prev} := the previous temperature of processor P_i;
3: calculate the local temperature increment of P_i after Δt by
$I_i^{in} = \frac{T_i^{curr} - T_i^{prev}}{t_i^{curr} - t_i^{prev}} \cdot \Delta t$;
4: calculate P_i's neighbors effect I_i^{nb} based on equation (7)
5: x = determine the location of processor, corner, boundary or middle;
6: determine the weight of τ_i under mode x such that
$w_i^x = [\alpha_i^x \ \beta_i^x \ \gamma_i^x]$;
7: predict the future temperature of P_i by
$T_i^{pred} = w_i^x \cdot B_i$
where $B_i = [T_i^{curr} \ I_i^{in} \ I_i^{nb}]$;

rithm to reduce the complexity for temperature prediction by applying the least-square estimation [30] to derive one uniform and normal weight matrix for all three different neighbor cases.

For any task τ_i, from equation (8), we know that the temperature prediction problem is formulated by

$$T_i^{pred} = \alpha_i \cdot T_i^{curr} + \beta_i \cdot \mathcal{I}_i^{in} + \gamma_i \cdot \mathcal{I}_i^{nb}.$$

To map the above temperature prediction problem into a general least-square problem, we construct a linear model for the output T^{pred} by the following linear parameterized expression

$$T^{pred}(t) = \alpha \cdot T^{curr}(t) + \beta \cdot I^{in}(t) + \gamma \cdot I^{nb}(t),$$

where $t = [t_1, t_2, t_3]$ is the model's input vector, $T^{curr}(t)$, $I^{in}(t)$ and $I^{nb}(t)$ are known functions of t, and α, β and γ are unknown parameters to be estimated. Let $\hat{T}$ represent $[T^{curr}(t), I^{in}(t), I^{nb}(t)]$, and $\hat{W}$ represent $[\alpha, \ \beta, \ \gamma]$. In our model, let t be time units, and can be chosen from three different scenarios with respect of neighbor processor condition, i.e. $t \in [t^c, t^b, t^m]$, where t^c, t^b, t^m represent the scenarios for corner, boundary and middle processor respectively.

To identify the unknown parameters, $\hat{W}$, experiments usually have to obtain a training data set $(T_j^{pred}(t); T_j^{curr}(t), I_j^{in}(t), I_j^{nb}(t))$, where $j = 1, ..., n$. Expressed in matrix notation, the following equation can be obtained:

$$\hat{T}^{pred} = \hat{T} \ \times \hat{W} \ ,$$

where $\hat{T}$ is a 3×3 matrix:

$$\hat{T} = \begin{bmatrix} T^{curr}(t^c) & I^{in}(t^c) & I^{nb}(t^c) \\ T^{curr}(t^b) & I^{in}(t^b) & I^{nb}(t^b) \\ T^{curr}(t^m) & I^{in}(t^m) & I^{nb}(t^m) \end{bmatrix} \tag{12}$$

$\hat{W}$ is a 3×1 unknown weight parameter vector:

$$\hat{W} = [\alpha,\ \beta,\ \gamma]^T, \tag{13}$$

and $\hat{T}^{pred}$ is a 3×1 output vector:

$$\hat{T}^{pred} = [T^c,\ T^b,\ T^m]^T. \tag{14}$$

If $(\hat{T}^{pred})^T \hat{T}^{pred}$ is nonsingular, the least square estimator can be derived as

$$\hat{W} = (\hat{T}^T \hat{T})^{-1} \hat{T}^T \hat{T}^{pred}. \tag{15}$$

Eventually, we predict the future temperature by applying equation (8), with the corresponding task-based weight parameter obtained by equation (15).

Algorithm 3 Neighbor-Normalized Temperature Prediction

1: T_i^{curr} := the current temperature of processor P_i;
2: T_i^{prev} := the previous temperature of processor P_i;
3: calculate the local temperature increment of P_i after Δt by
$I_i^{in} = \frac{T_i^{curr} - T_i^{prev}}{t_i^{curr} - t_i^{prev}} \cdot \Delta t$;
4: calculate P_i's neighbors effect I_i^{nb} based on equation (7)
5: get the weight parameter w_i for current task, $w_i^x = [\alpha_i^x, \beta_i^x, \gamma_i^x]$;
6: predict the future temperature of P_i by
$T_i^{pred} = w_i \cdot B_i$
where $B_i = [T_i^{current}, I_i^{in}, I_i^{nb}]$;

The *NNTP* prediction algorithm could be described in the similar expression as algorithm 2. After calculating the local temperature increment and the neighbor effect, the weight parameters of the current task can be calculated by using the least-square estimation method. The future temperature is predicted by applying equation (7).

5 PROACTIVE DTM ALGORITHM

With the temperature prediction technique, we introduce our complete proactive thermal management algorithm in this section.

5.1 Candidate processor for migration

When the thermal emergency is detected by the temperature prediction technique, one solution is migrating the task away from the hot processor to bring down the temperature. To identify the appropriate destination, one common approach [13] is to migrate the task to the processor with the lowest current temperature. However, selecting the coolest processor is not always the best decision. Due to the sudden neighboring processor temperature change or the potential of the big temperature increasing rate by itself, the coolest core can rapidly become a hotspot after the next sampling interval. Thus, to address this problem in our approach, besides the current temperature of the candidate processor, we consider its

neighboring temperatures, as well as its temperature changing rate to make the migration decision.

We first introduce a concept, *heat index*, to quantify how hot a candidate processor (i.e. P_k) is.

Definition 5.1. *Given processor P_k, the* heat index *of P_k, denoted as $\mathcal{H}(P_k)$, is defined as*

$$\mathcal{H}(P_k) = \frac{\sum_{P_j \in \mathcal{P}_k^{NB} \bigcup \{P_k\}} T_j}{|\mathcal{P}_k^{NB} \bigcup \{P_k\}|}. \tag{16}$$

Intuitively, the smaller the heat index of a processor is, the better the candidate processor it can be.

Besides the heat index of a processor, we also consider the temperature changing rates of itself as well as its neighbors. We present the following definition, i.e. the *heat index increasing factor* of a processor P_k, to capture this concept.

Definition 5.2. *Given processor P_k, the* heat index increasing factor *of P_k, denoted as $\mathcal{I}(P_k)$, is defined as*

$$\mathcal{I}(P_k) = \frac{\sum_{P_j \in \mathcal{P}_k^{NB} \bigcup \{P_k\}} \frac{T_j^{curr} - T_j^{prev}}{t_j^{curr} - t_j^{prev}}}{|\mathcal{P}_k^{NB} \bigcup \{P_k\}|}. \tag{17}$$

According to Definition 5.2, $\mathcal{I}(P_k)$ indicates how fast the temperature at P_k and its neighbors can increase in average. Thus, the smaller the heat index increasing factor, the better the candidate processor can be. From equation (16) and (17), we choose the migration candidate as the one that minimizes

$$\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t, \tag{18}$$

where Δt is the length of the sampling interval.

Note that task migration is not always effective in dealing with a thermal emergency, especially when the workload is heavy. Given a processor P_k in thermal emergency, it does not help much if the selected target processor (e.g. P_k) for migration has a temperature very close to the peak temperature limit, even if the $\mathcal{H}(P_k)+\mathcal{I}(P_k)\cdot\Delta t$ is minimum among all other processors. Besides, too many unnecessary task migrations may cause redundant context switch overhead, which could degrade throughput performance. To avoid this scenario, in our approach, the tasks on processor P_k are only allowed to migrate to processor P_k if

$$\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t \leq T^{THRESHOLD}, \tag{19}$$

where $T^{THRESHOLD}$ is the given temperature constraint. Otherwise, we can adopt an alternative solution to cool down the processor. Such as selecting a safe working speed for the processor P_k by using the same offline thermal profiling analysis approach as presented before.

5.2 Thermal management algorithm

In this subsection, we introduce our proposed thermal management algorithm, the Neighbor-Aware Dynamic Thermal Management (NADTM) algorithm, to maximize the throughput

of a multi-core system while keeping the temperature under a predefined peak temperature limit.

Algorithm 4 Neighbor-Aware Dynamic Thermal Management (NADTM) Algorithm

1: $T_i^{prev} := T_i^{curr}$ // the temperature at previous sampling point ;
2: $T_i^{curr} :=$ the temperature of P_i from temperature sensor;
3: $T_i^{pred} :=$ predicted temperature of P_i at next sampling point based on equation (8);
4: **if** $T_i^{pred} > T^{THRESHOLD}$ **then**
5: $P_k :=$ the processor from $\mathcal{P}$ such that $\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t$ is minimum;
6: **if** $\mathcal{H}(P_k) + \mathcal{I}(P_k) \cdot \Delta t \leq T^{THRESHOLD}$ **then**
7: migrate current running tasks on P_i to P_k;
8: **else**
9: degrade the performance of P_i by setting its speed to the pre-defined safe speed (i.e S_i^{SAFE});
10: **end if**
11: **end if**

The NADTM algorithm is presented in Algorithm 4. For processor P_i, we read the temperature sensor to get its current temperature and then predict its temperature at the next sampling point based on the method described in section 4. If the predicted temperature exceeds the temperature constraint, we will search for a candidate processor that we can migrate the tasks to. The candidate processor is selected based on method presented in section 5.1. If such a processor is not available, we select a safe speed from the thermal profile lookup table as discussed in Section 3.2.

We assume that the weights in equation (8) have been identified off-line. The safe speed to run a processor is essentially the maximum processor speed for a processor such that its peak temperature will not exceed the temperature constraint. We also assume that this speed is obtained off line.

6 EXPERIMENTAL RESULTS

In this section, we first introduce the experiment setup. Then we validate the accuracy of our neighbor-aware temperature prediction technique by comparing it with the enhanced reactive approach. Finally, we analyze the performance improvement of our NADTM algorithms.

6.1 Experiment setup

An overview of the practical desktop environment is depicted in Figure 6. The target platform is a Dell Precision T1500 desktop workstation with an Intel i5 750 quad core microprocessor, which running Linux operating system with kernel version of 2.6.32. The Intel i5 microprocessor has integrated with *Enhanced Intel SpeedStep Technology* (EIST) [1] and supports 12 different working frequency levels ranging from 1.2GHz to 2.66GHz. We adopted the *CPUfreq* Linux kernel subsystem to implement the DVFS features. Furthermore, we implement task migration technique by changing the *CPU affinity* of the running process.

A Dell Precision T1500 desktop workstation has two cooling components: the heat sink and cooling fans. The cooling effect of the heat sink depends on its physical characteristic,

which does not change when running an application. The fans, on the other hand, can be adjusted dynamically between the maximum of 4500RPM (rotation per minute) and minimum of 1500RPM. The fan speed is fixed at the minimal speed to ensure all experiments were conducted under the same cooling condition.

A key aspect in our study is to capture the temperature information accurately and timely. Dell Precision T1500 desktop has an external thermal sensor, located underneath the CPU chip. A more accurate method is to read the temperature value directly from the built-in digital thermal sensor integrated with each core. In our experiment, we used the on-chip thermal sensors to measure the instant temperature of the processor. The resolution of the on-chip thermal sensor is only 1^oC; and the minimal time for a temperature sensor to reflect a change in temperature is approximately 1 second [2]. To ease our implementation and tests, we adopted a Linux hardware monitoring tool called *Lm-sensors* to capture the temperature, to set the fan speed, to vary supply voltage and working frequency.

All experiments were carried out with the same ambient temperature. We selected six benchmarks *galgel, parser, ammp, crafty, lucas* and *equake* from the well-known commercial benchmark SPEC CPU2000, including both integer and floating point operation to obtain credible and comparable experiment results. Those benchmarks have been grouped into three categories, which are *hot*, *warm*, and *cool*, based on their thermal characteristics. To build up the temperature lookup table, we conducted the off-line thermal profiling analysis by running each benchmark at different CPU speeds. The stable temperatures with their corresponding speed levels were stored in a lookup table. To ensure the schedule effectiveness, each benchmark was tested with the *hot* benchmark applications running on its neighboring processors. In the lookup table, the safe speed is the maximal speed corresponding to the stable temperature lower than the given temperature constraint.

6.2 Prediction analysis

To evaluate the accuracy of our NADTM temperature prediction technique, we compared our heuristic with the conventional temperature prediction approach, which uses the previous and current temperature values of a processor to predict the next temperature value without considering the heat transfer from the neighboring processors.

Figure 7 shows the temperature traces of running benchmark *galgel*, as well as the temperature prediction results based on our proposed temperature prediction method and the conventional one. From Figure 7, we can clearly see that the temperature prediction results of using the NADTM approach are much closer to the actual temperature values than the conventional approach. Also, the NADTM approach has a smaller maximum prediction error of 1^oC comparing with 3^oC by the conventional approach. The results shown in Figure 7 demonstrate that, by taking consideration of the heat transfer impacts from the neighboring processors, the temperature prediction methods introduced in section 4 can achieve a higher accuracy than the traditional method.

To further validate this conclusion, we ran different benchmark programs on our test platform. First, temperature prediction results were collected and compared with the actual temperature value. Then the temperature prediction accuracy, using two different prediction methods, was plotted in Figure 8. (i.e., the prediction accuracy is the number of accurate predictions over total number of predictions). In order to compare the two approaches, both

results are normalized to the approach without NADTM. From Figure 8 we can see that our NADTM approach can improve the temperature prediction accuracy by 38% in average compared to the conventional approach. Based on those experiment results, our neighboring aware temperature prediction technique could effectively improve the prediction accuracy.

6.3 Performance analysis

Since our prediction technique can effectively detect the thermal emergency, we further evaluate the performance of our NADTM algorithm in term of its capability to satisfy the temperature constraints and to maximize system throughput by comparing with the conventional reactive approach presented in Section 2.1, and the ERDTM algorithm presented in Section 3.

Ideally, a thermal management schedule should push the computing system to the thermal boundary while delivering the highest system performance. At the same time, the temperature needs to be carefully maintained under the threshold. Thus, we implement each of the dynamic thermal management algorithms to control the processor temperature under a pre-defined threshold, (i.e. 55^oC in this work).

After running the conventional reactive algorithm, we can see that the processor temperature violates the given threshold frequently and can exceed the temperature threshold as much as 4^oC as shown in Figure 9. This is mainly due to two reasons: first, the thermal sensor cannot keep up with the temperature changes timely; second, the algorithm either decreases or increases the speed whenever the temperature change occur, It actually overreacts and misleads the speed setting of the processor before the temperature become stable. However, the Figure 10 shows that after we implement the enhanced ERDTM algorithm, the number of temperature violations have been significantly reduced, and the temperature oscillation range has been reduce by 50%. This is because the non-constant sampling periods can catch the temperature changes timely. On the other hand, the offline thermal profile lookup table could quickly locate the proper speed level that can satisfy the temperature constraint and dramatically reduce the redundant frequency-switching overhead. However, there are still several spikes (i.e. temperature violations) over the ERDTM algorithm. This is due to the drawback of the reactive approach itself. For example, it cannot take action until the sensor detects the thermal emergency. In contrast, our proactive NADTM algorithm can safely predict the thermal emergency as proved in the previous subsection and provide enough time for the thermal management algorithm to react the temperature change. As shown in Figure 11, our NADTM algorithm can completely eliminate the temperature violations and let the computing system deliver maximal throughput. We analyze the throughput performance of our proposed NADTM algorithm in the next subsection.

6.4 Throughput analysis

To analyze the throughput of our NADTM algorithm, we only compare it to the proactive approaches. This is due to two reasons: first, in the previous subsection, we already proved that reactive approach cannot affectively management the processor under the temperature constraint. And, preventing thermal violation is the first priority for thermal-aware scheduling algorithm, thus any thermal violation is not acceptable for a thermal-aware algorithm. Second, because of the close correlation between chip temperature and working speed, the

reactive approach has longer time to make the system temperature over the threshold, which will result in a higher throughput. Thus, it would not be fair to compare our approach to the reactive approach, which cannot satisfy the temperature constraint.

We use *NP, CP* to denote neighbor-aware prediction and conventional prediction, and *NM, CM* for neighbor-aware migration and conventional migration, respectively. The conventional temperature prediction approach refers to the one that predicts the future temperature solely based on its own temperature history. And the conventional migration approach refers to the approach that simply migrates the running tasks from the hottest core to the coolest core. As a result, we have four combinations, *i.e. CP_CM NP_CM, CP_NM* and *NP_NM.*

We first compare the throughput of each approach when running a single task on our hardware platform. In this experiment, six previously used benchmarks have been selected to provide reliable experiment results. The execution times by using different approaches have been recorded for comparison, those experiment results have been normalized and plotted in Figure 12(a). The results show that, with the neighbor-aware prediction algorithm *i.e. NP_CM* can improve the throughput over *CP_CM* as much as 1.7% in average. Since our prediction technique is more accurate than the conventional approach as shown before, it helps to make better scheduling decision and thus improves the performance. Another observation is that *CP_NM* improves the throughput over *CP_CM* as much as 3.6%. This is because *CP_NM* can find the appropriate migration candidate rather than simply locate the coolest core. By combining our proposed prediction and task migration algorithm together, *NP_NM* can achieve an average of 5.8% overall throughput improvement when compared with *CP_CM.*

To further test our thermal management algorithm, we assigned multiple tasks to the multicore platform. By gradually increasing the number of tasks running on the multicore processor, their corresponding execution times have been recorded for comparison. The execution times have been normalized and plotted in Figure 12(b). As we can see from the experiment results, the overall throughput decreases as the number of tasks increases. Another important observation is that when the number of tasks is larger than the number of core (*i.e. the number of task is more than 4*), the throughput drops significantly. The experiment results show that the throughput for the *NP_CM* decreased by 0.9% while the tasks increased from 1 to 6. The throughput for *CP_NM* decreased by 3%. The throughput decreased by 3.6% for the overall NADTM algorithm. All these results show that the proposed algorithm works better with a lighter workload than a heavy workload.

7 CONCLUSION

In this paper, we first studied the limitations of the existing theoretical work. Then we developed a predictive thermal-aware algorithm for the practical multi-core platform to maximize the system throughput under peak temperature constraint. Our proposed approach takes the neighbor effect into consideration to make a more accurate temperature prediction and to determine a better migration destination. The algorithm has been validated on an Intel multi-core platform, and the experiment results illustrate that our NADTM algorithm can significantly improve the system throughput while satisfying the temperature constraint.

8 ACKNOWLEDGEMENT

This work is supported in part by NSF under projects CNS-0969013, CNS-0917021, CNS-1018108, and CNS-1249223.

REFERENCES

[1] Enhanced intel speedstep technology [available online]: http://www.intel.com/support/processors/sb/cs-028855.htm.

[2] Lm-sensors linux hardware monitoring: http://www.lm-sensors.org.

[3] Y. Ahn, Y.-S. Hwang, and K.-S. Chung. Flexible framework for dynamic management of multi-core systems. In *SoC Design Conference (ISOCC), 2009 International*, pages 237 –240, Busan, Korea, November 2009.

[4] P. Bailis, V. J. Reddi, S. Gandhi, D. Brooks, and M. Seltzer. Dimetrodon: processor-level preventive thermal management via idle cycle injection. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 89 –94, San Diego, USA, June 2011.

[5] T. Chantem, X. S. Hu, and R. P. Dick. Online work maximization under a peak temperature constraint. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '09, pages 105–110, New York, USA, 2009. ACM.

[6] T. Chen, J. Huang, L. Xiang, and Q. Shi. Dynamic power management framework for multi-core portable embedded system. In *Proceedings of the 1st International Forum on Next-generation Multicore/Manycore Technologies*, IFMT '08, pages 1:1–1:4, New York, USA, 2008. ACM.

[7] S. W. Chung and K. Skadron. Using on-chip event counters for high-resolution, real-time temperature measurement. In *Thermal and Thermomechanical Phenomena in Electronics Systems, ITHERM '06. The Tenth Intersociety Conference on*, pages 114 –120, June 2006.

[8] A. Cohen, F. Finkelstein, A. Mendelson, R. Ronen, and D. Rudoy. On estimating optimal performance of CPU dynamic thermal management. *IEEE Comput. Archit. Lett.*, 2:6–, January 2003.

[9] A. K. Coskun, T. S. Rosing, and K. C. Gross. Proactive temperature management in MPSoCs. In *Proceeding of the 13th International Symposium on Low Power Electronics and Design*, ISLPED '08, pages 165–170, New York, USA, 2008. ACM.

[10] A. K. Coskun, T. S. Rosing, and K. Whisnant. Temperature aware task scheduling in MPSoCs. In *Proceedings of the conference on Design, Automation and Test in Europe*, DATE '07, pages 1659–1664, San Jose, USA, 2007. EDA Consortium.

[11] A. K. Coskun, R. Strong, D. M. Tullsen, and T. Simunic Rosing. Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *Proceedings of the eleventh International Joint Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '09, pages 169–180, New York, USA, 2009. ACM.

[12] Y. Ge, P. Malani, and Q. Qiu. Distributed task migration for thermal management in many-core systems. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 579 –584, Anaheim, USA, June 2010.

[13] M. Gomaa, M. D. Powell, and T. N. Vijaykumar. Heat-and-run: leveraging SMT and CMP to manage power density through the operating system. *SIGOPS Oper. Syst. Rev.*, 38:260–270, October 2004.

[14] C. Isci and M. Martonosi. Runtime power monitoring in high-end processors: methodology and empirical data. In *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 36, page 93, Washington, USA, 2003. IEEE Computer Society.

[15] O. Khan and S. Kundu. Hardware/software co-design architecture for thermal management of chip multiprocessors. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 952 –957, Dresden, Germany, April 2009.

[16] W. Kim, M. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core DVFS using on-chip switching regulators. In *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 123 –134, Salt Lake City, USA, February 2008.

[17] J. S. Lee, K. Skadron, and S. W. Chung. Predictive temperature-aware DVFS. *Computers, IEEE Transactions on*, 59(1):127 –133, January 2010.

[18] J. Li, M. Qiu, J. Niu, T. Chen, and Y. Zhu. Real-time constrained task scheduling in 3D chip multiprocessor to reduce peak temperature. In *Proceedings of the 2010 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing*, EUC '10, pages 170–176, Washington, USA, 2010. IEEE Computer Society.

[19] G. Liu, M. Fan, and G. Quan. Neighbor-aware dynamic thermal management for multi-core platform. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 187 –192, Dresden, Germany, March 2012.

[20] G. Liu and G. Quan. Thermal aware scheduling on an Intel desktop computer. In *Southeastcon, 2011 Proceedings of IEEE*, pages 79 –84, Nashville, USA, March 2011.

[21] G. Liu, G. Quan, and M. Qiu. Throughput maximization for Intel desktop platform under the maximum temperature constraint. In *Green Computing and Communications (GreenCom), 2011 IEEE/ACM International Conference on*, pages 9 –15, Chengdu, China, August 2011.

[22] F. Mulas, D. Atienza, A. Acquaviva, S. Carta, L. Benini, and G. De Micheli. Thermal balancing policy for multiprocessor stream computing platforms. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(12):1870 –1882, December 2009.

[23] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, and D. Atienza. Thermal balancing policy for streaming computing on multiprocessor architectures. In *Proceedings of the conference on Design, Automation and Test in Europe*, DATE '08, pages 734–739, New York, USA, 2008. ACM.

[24] M. Qiu, L. T. Yang, Z. Shao, and E. H.-M. Sha. Dynamic and leakage energy minimization with soft real-time loop scheduling and voltage assignment. *IEEE Trans. Very Large Scale Integr. Syst.*, 18(3):501–504, March 2010.

[25] E. Rohou and M. D. Smith. Dynamically managing processor temperature and power. In *In 2nd Workshop on Feedback-Directed Optimization*, 1999.

[26] S. Wang and R. Bettati. Reactive speed control in temperature-constrained real-time systems. In *Real-Time Systems, 2006. 18th Euromicro Conference on*, pages 10 pp. –170, Dresden, Germany, 2006.

[27] Y. Wang, K. Ma, and X. Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, pages 314–324, New York, USA, 2009. ACM.

[28] T. Wei, X. Chen, and P. Mishra. Designing a multi-core hard real-time test bed for energy measurement experiments. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1998–1999, New York, USA, 2009. ACM.

[29] I. Yeo and E. J. Kim. Temperature-aware scheduler based on thermal behavior grouping in multicore systems. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 946 –951, Nice, France, April 2009.

[30] I. Yeo, C. C. Liu, and E. J. Kim. Predictive dynamic thermal management for multicore systems. In *Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE*, pages 734 –739, Anaheim, USA, June 2008.

[31] S. Zhang and K. Chatha. Approximation algorithm for the temperature-aware scheduling problem. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 281 –288, San Jose, USA, November 2007.

[32] S. Zhang and K. S. Chatha. Thermal aware task sequencing on embedded processors. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 585–590, New York, USA, 2010. ACM.

FIGURES AND TABLES

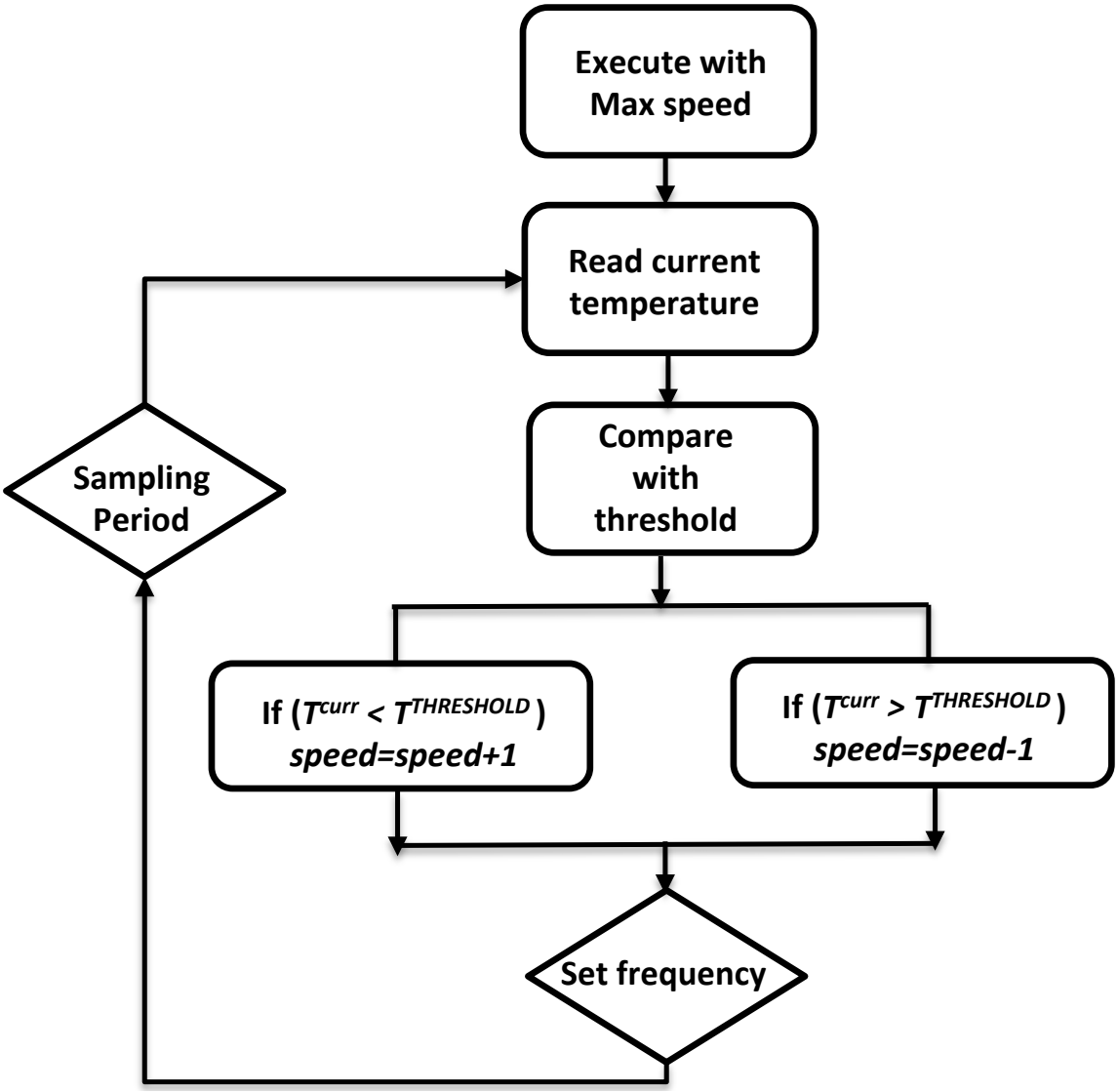


Figure 1: The conventional dynamic approach for throughput maximization.

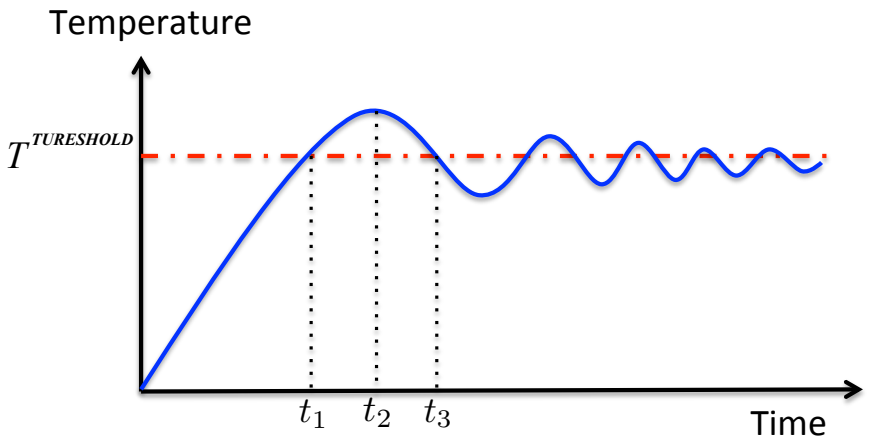


Figure 2: An example of temperature trace.

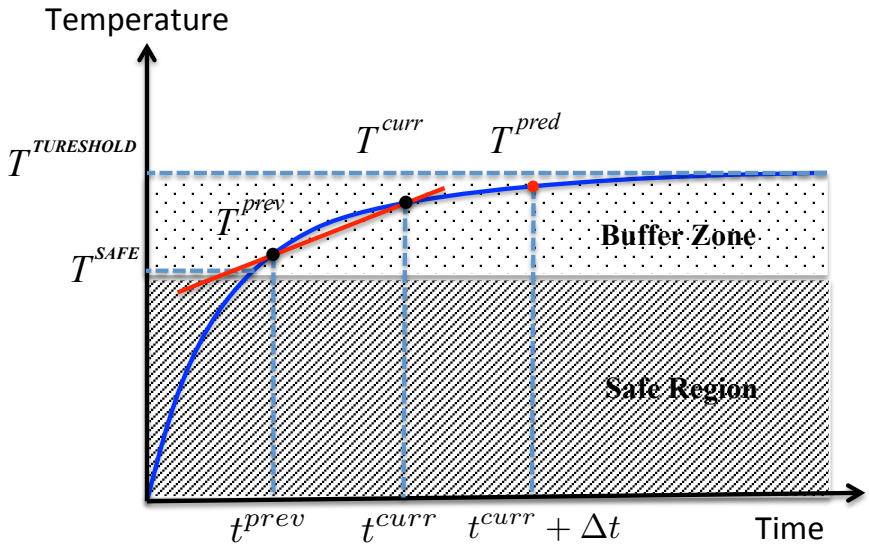


Figure 3: Temperature history based prediction

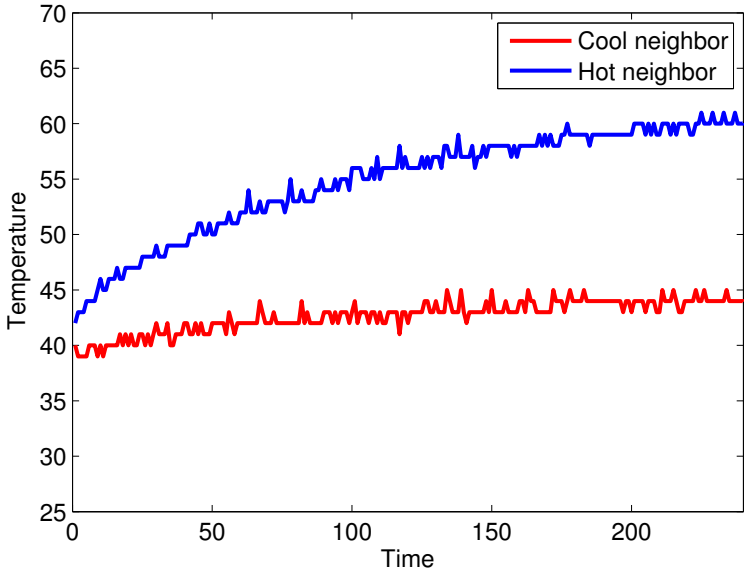


Figure 4: Temperature trace with Hot and Cool neighbor processors

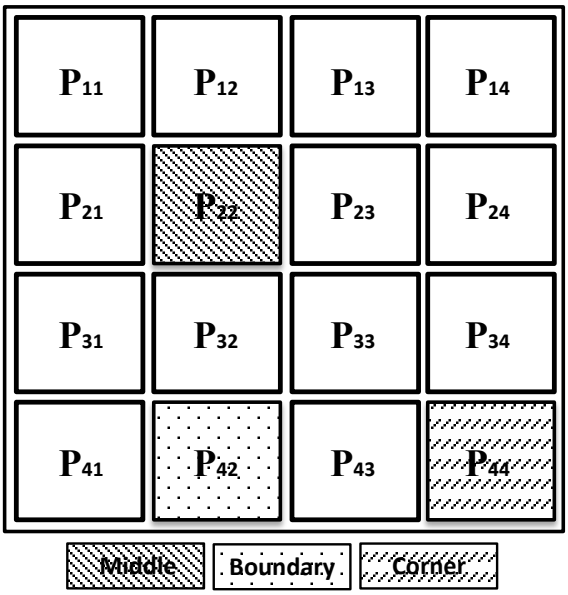


Figure 5: Different processor location scenarios

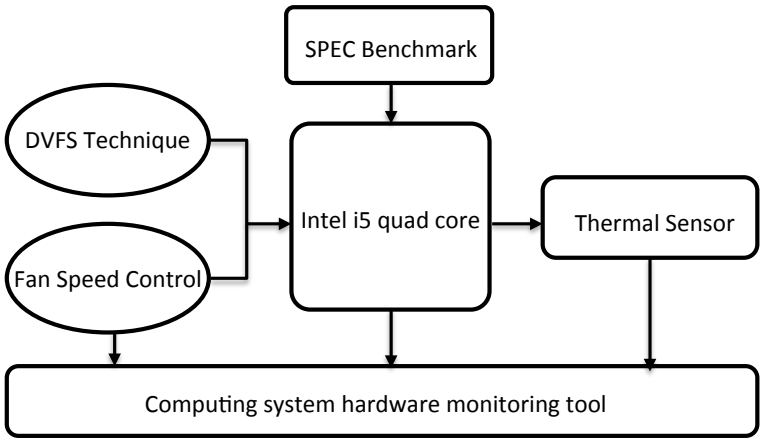


Figure 6: Structure of hardware platform

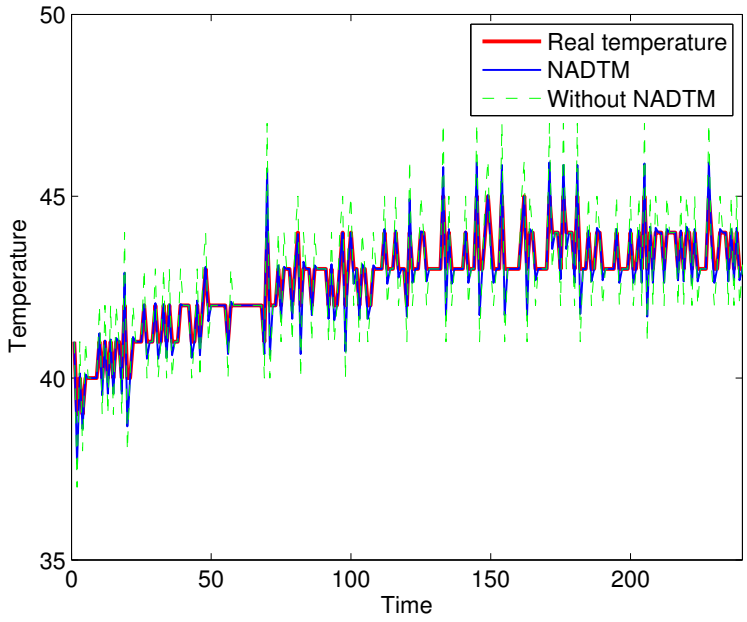


Figure 7: NADTM compares to the conventional prediction method, which does not take the neighbor effect into consideration

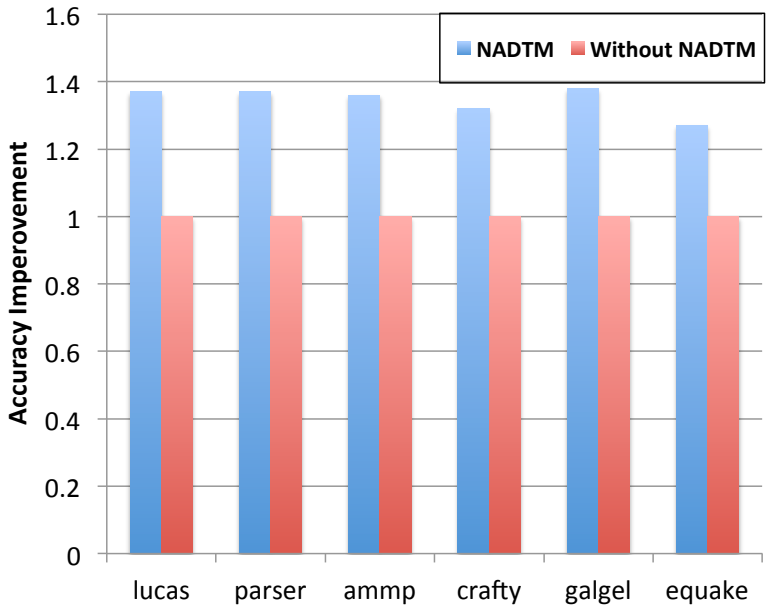


Figure 8: Prediction accuracy comparison with different benchmarks

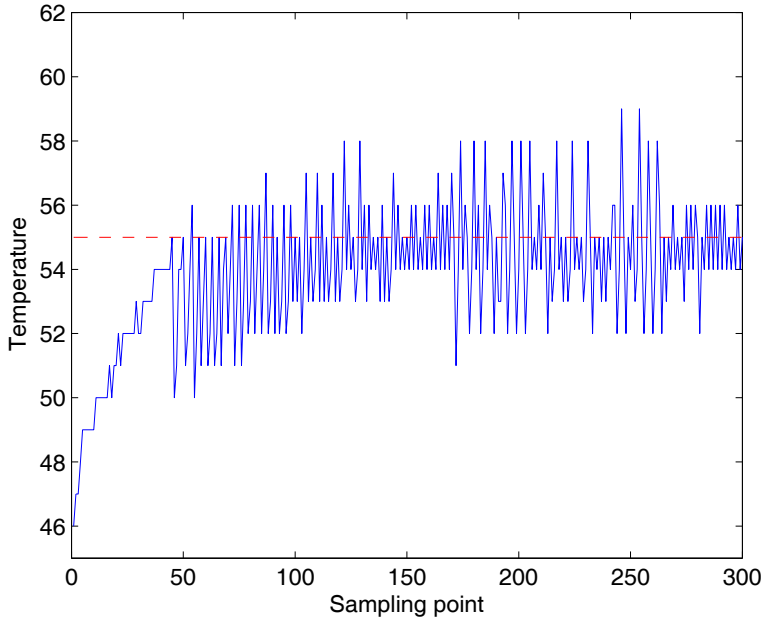


Figure 9: Massive temperature violations occur after implement the conventional reactive approach.

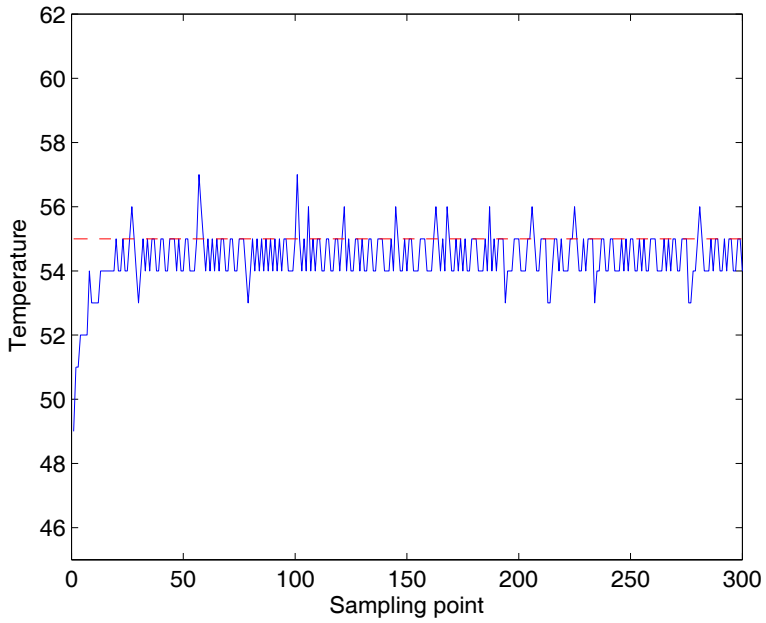


Figure 10: Temperature violations has been significantly reduced after using the ERDTM algorithm.

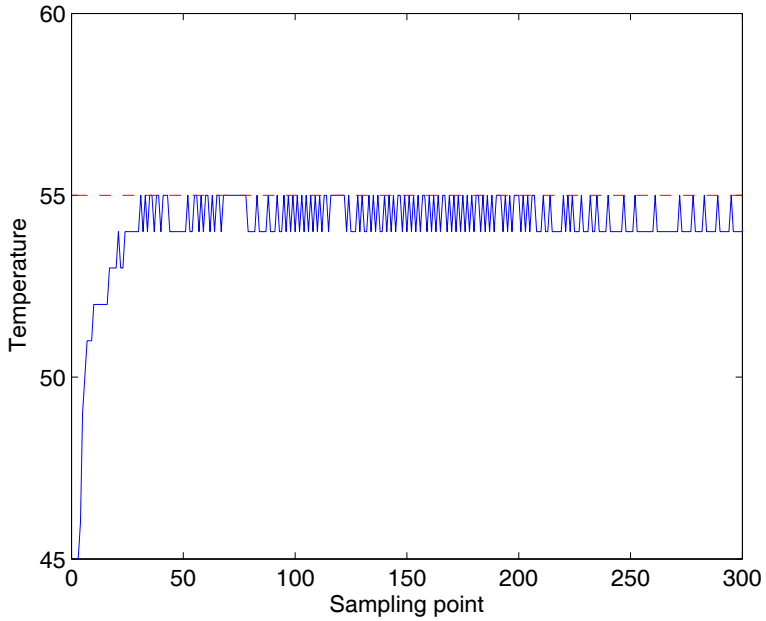
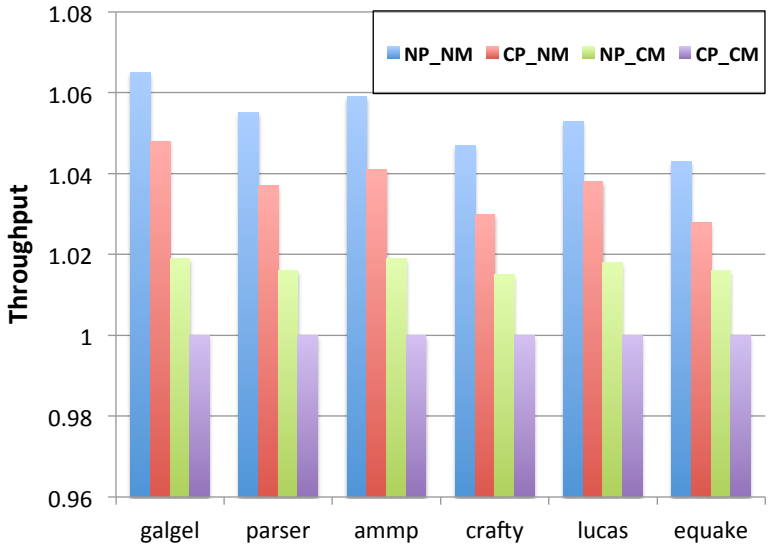
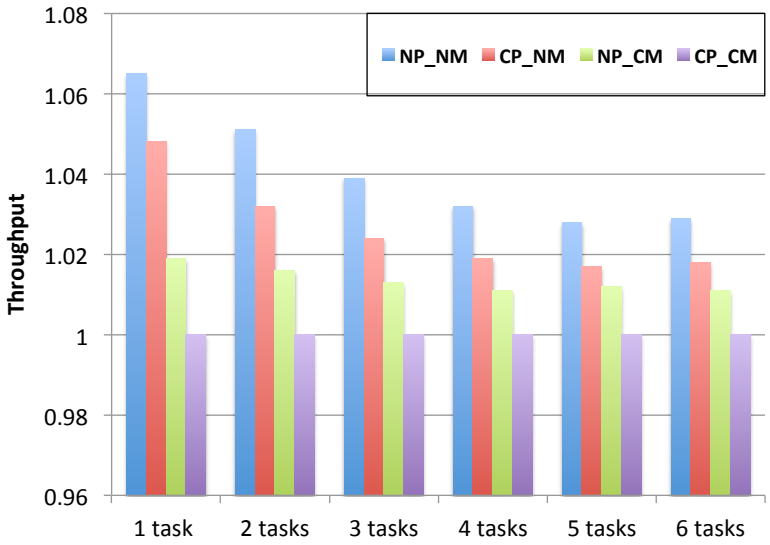


Figure 11: The proposed NADTM algorithm could completely eliminate the temperature violation.



(a) Throughput comparison by using different single task



(b) Throughput comparison with multiple tasks running on the multiprocessor platform

Figure 12: Throughput comparison with four different approaches. *NP and CP represent the neighbor-aware and conventional prediction respectively. NM and CM represent neighbor-aware and conventional migration respectively*

BIOGRAPHIES

Guanglei Liu is currently a masters student in the Department of Computer Engineering at San Jose State University, California. He received his Ph.D. degree in Electrical and Computer Engineering from Florida International University, Miami in August 2012, and his Bachelors degree in Electrical Engineering from Harbin University, Harbin, China in 2006. He is a student member of IEEE and his research interests include thermal and power-aware computing and embedded real-time operating system design.

Ming Fan is a PhD candidate in the Department of Electrical and Computer Engineering at the Florida International University, FL. He received both the BS and MS degrees in computer engineering from Bei Hang University, Beijing, China, in 2006 and 2009, respectively. His research interests include real-time systems, power and thermal-aware computing and fault-tolerant systems.

Gang Quan is currently an Associate Professor in Electrical and Computer Engineering Department, Florida International University (FIU), Miami. He received the B.S. degree from the Tsinghua University, Beijing, China, the M.S. degree from the Chinese Academy of Sciences, Beijing, and the Ph.D. degree from the University of Notre Dame, Notre Dame, IN. Before joining FIU, he was an assistant professor at the Department of Computer Science and Engineering, University of South Carolina. His research interests includes real-time system, power/thermal aware design, embedded system design, advanced computer architecture. Prof. Quan received the NSF CAREER award in 2006 and the Best Paper Award from the Design Automation Conference (DAC)

Meikang Qiu is currently an Assistant Professor in Electrical and Computer Engineering Department, University of Kentucky, Lexington. He received the BE and ME degrees from Shanghai Jiao Tong University, China. He received the MS and PhD degrees of computer science from the University of Texas at Dallas, in 2003 and 2007, respectively. He had worked at Chinese Helicopter RD Institute and IBM. He has published more than 150 peer-reviewed papers, including 60 journal papers. He has been on various chairs and TPC members for many international conferences. He served as the Program Chair of IEEE EmbeddCom 09 and EM-Com 09. He is the recipient of the ACM Transactions on Design Automation of Electronic Systems (TODAES) 2011 Best Paper Award. He received Navy Summer Faculty Award in 2012 and Air Force Summer Faculty Award 2009. He won four best paper awards (IEEE Embedded and ubiquitous Computing (EUC 09), IEEE/ACM GreenCom 10, IEEE CSE 10, IEEE ICESS 12) and one best paper nomination. He also holds 2 patents and has published 3 books. His research is supported by NSF. His research interests include embedded systems, computer security, and wireless sensor networks. He is a senior member of the IEEE.