Hierarchical Dynamic Thermal Management Method for High-Performance Many-Core Microprocessors

HAI WANG and JIAN MA, University of Electronic Science and Technology of China
SHELDON X.-D. TAN, University of California at Riverside
CHI ZHANG and HE TANG, University of Electronic Science and Technology of China
KEHENG HUANG and ZHENGHONG ZHANG, Southwest China Research Institute of Electronic Equipment

It is challenging to manage the thermal behavior of many-core microprocessors while still keeping them running at high performance since the control complexity increases as the core number increases. In this article, a novel hierarchical dynamic thermal management method is proposed to overcome this challenge. The new method employs model predictive control (MPC) with task migration and a DVFS scheme to ensure smooth control behavior and negligible computing performance sacrifice. In order to be scalable to many-core systems, the hierarchical control scheme is designed with two levels. At the lower level, the cores are spatially clustered into blocks, and local task migration is used to match current power distribution with the optimal distribution calculated by MPC. At the upper level, global task migration is used with the unmatched powers from the lower level. A modified iterative minimum cut algorithm is used to assist the task migration decision making if the power number is large at the upper level. Finally, DVFS is applied to regulate the remaining unmatched powers. Experiments show that the new method outperforms existing methods and is very scalable to manage many-core microprocessors with small performance degradation.

CCS Concepts: • **Hardware → Temperature control**

Additional Key Words and Phrases: Dynamic thermal management, many-core microprocessor, model predictive control, DVFS, task migration

ACM Reference Format:
Hai Wang, Jian Ma, Sheldon X.-D. Tan, Chi Zhang, He Tang, Keheng Huang, and Zhenghong Zhang. 2016. Hierarchical dynamic thermal management method for high-performance many-core microprocessors. ACM Trans. Des. Autom. Electron. Syst. 22, 1, Article 1 (July 2016), 21 pages.
DOI: http://dx.doi.org/10.1145/2891409

This work is supported in part by the National Natural Science Foundation of China under Grant No. 61404024; in part by the Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry; in part by the Open Foundation of State Key Laboratory of Electronic Thin Films and Integrated Devices under Grant No. KFJJ201409; in part by the National Science Foundation (NSF) under Grant No. CCF-1255899; in part by the Semiconductor Research Corporation (SRC) under Grant No. 2013-TJ-2417; and in part by the National Natural Science Foundation of China under Grant No. 61204031.
Authors' addresses: H. Wang, J. Ma, C. Zhang, and H. Tang, School of Microelectronics and Solid-State Electronics, University of Electronic Science and Technology of China, No.4, Sec. 2, North Jianshe Rd., Chengdu 610054, China; emails: wanghai@uestc.edu.cn, majian@std.uestc.edu.cn, zhangc@uestc.edu.cn, tanghe@uestc.edu.cn; S. X.-D. Tan, Department of Electrical and Computer Engineering, University of California at Riverside, 900 University Ave., Riverside, California 92521; email: stan@ece.ucr.edu; K. Huang and Z. Zhang, Southwest China Research Institute of Electronic Equipment, No. 496 West Yingkang Rd., Chengdu 610036, China; emails: {39774128, 1274006713}@qq.com.

DOI: http://dx.doi.org/10.1145/2891409

1. INTRODUCTION

Extremely high operating temperatures have negative effects on the reliability of a microprocessor. The increasing power density and spatial power variation in the new-generation high-performance multi/many-core microprocessors introduce severe local hot spot problems, resulting in performance degradation, high cooling costs, and serious reliability issues. Finding economical and efficient methods to solve the high temperature issue and improve both performance and reliability for multi/many-core microprocessors remains a challenging task [Brooks et al. 2007].

The dynamic thermal management (DTM) method is one effective technique to improve the thermally related performance of microprocessors [Donald and Martonosi 2006]. It controls the running behavior of the microprocessor to make sure its temperature is within the safe range while keeping its computing performance at a high level. DTM is usually performed by two categories of methods: task migration and DVFS. The task migration method switches tasks among different cores in multi/many-core microprocessors to lower the peak temperature of the chip [Powell et al. 2004; Ge et al. 2010; Chantem et al. 2011; Liu et al. 2012; Ayoub and Rosing 2009; Ebi et al. 2009] and is also used to lower the energy consumption in heterogenous multicore systems [Cong and Yuan 2012]. Task migration usually leads to high throughput of the system since all cores are running at the maximum speed. But it may suffer from a high average temperature problem without any other DTM methods being involved.

The DVFS method [Skadron et al. 2003; Jayaseelan and Mitra 2009; Mutapcic et al. 2009] controls voltage and operating frequency speed to adjust the heat dissipation of the chip. Recently, DVFS was also used in the dark silicon scenario, which determines the V/f levels for dark silicon chips with temperature as the constraint [Khdr et al. 2015; Muthukaruppan et al. 2013]. DVFS is able to guarantee the thermal safety of the chip and save energy of the chip, but computing performance of the microprocessor is sacrificed due to the frequency scaling.

In order to guide the DVFS and/or task migration, a control scheme is usually employed by DTM methods except for experience-based or heuristic-based ones. The controller uses a thermal model, thermal sensor information, and so forth and outputs the action guidance (e.g., how much frequency should be adjusted for DVFS) for DTM techniques. Many DTM methods are based on traditional control methods [Kadin et al. 2009], but these methods do not suit very well multi/many-core thermal systems due to their complex multimodal dynamics [Bartolini et al. 2013]. Recently, model predictive control (MPC) was introduced in DTM [Zanini et al. 2009; Wang et al. 2009; Bartolini et al. 2013]. It uses the thermal model of the microprocessor and outputs management suggestions on the power side. Since it makes predictions on the thermal behavior to enhance control efficiency, it is able to provide more accurate and effective management suggestions. Significant throughput improvement is reported by using MPC compared with traditional control methods [Bartolini et al. 2013].

Combining MPC with both DVFS and task migration may take the advantages from all three methods: the high quality control from MPC, good computing performance from task migration, and guaranteed thermal safety from DVFS. There are a number of works that combine two techniques out of the three. Specifically, works that combine task migration and DVFS include the experience-based method [Brooks and Martonosi 2001], the hybrid method that optimizes performance [Hanumaiah et al. 2011], and the hybrid method that optimizes energy consumption [Hanumaiah and Vrudhula 2014; Tan et al. 2015]. Works that contain both MPC and DVFS are presented in Zanini et al. [2009], Wang et al. [2009], and Bartolini et al. [2013]. However, combining MPC with task migration is much harder than combining MPC with DVFS. Recently, a work that combines all three methods was proposed [Ma et al. 2014]. However, this method is

only applicable to multicore microprocessors due to the high overhead introduced in integrating MPC and task migration, especially for many-core systems.

In this article, a new hierarchical dynamic thermal management method is proposed for high-performance many-core microprocessors. The new method uses model predictive control to guide the management process with both task migration and DVFS. In order to solve the scalability problem of performing MPC-based task migration on many-core systems, the new method makes the task migration decision at two levels. At the lower level, spatially adjacent cores are clustered into blocks, bipartite matching is performed on the powers of cores inside each block to make the in-block migration decision, and the unmatched powers are collected at the upper level for the second-round-migration decision making. A modified iterative minimum cut algorithm is introduced to speed up the decision-making process at the upper level. The new hierarchical method is highly scalable for many-core microprocessors with little overhead and is able to maintain the high performance of the chip without violating the thermal constraint.

2. MODEL-PREDICTIVE-CONTROL-BASED DYNAMIC THERMAL MANAGEMENT

In this section, we will introduce the MPC-based DTM method. The thermal model integrated in MPC is presented first in Section 2.1. How to compute the desired power using MPC in order to guide the DTM process is given next in Section 2.2. Finally, Section 2.3 shows the steps to perform task migration and DVFS based on the desired power from MPC.

2.1. Thermal Model of the Microprocessor

Due to the well-known duality between the thermal system and electrical circuit system, we can build the thermal model of the microprocessor using thermal equivalent resistors (thermal resistors), thermal equivalent capacitance (thermal capacitors), and thermal equivalent independent current and voltage sources. As a result, similar to the electrical system, the thermal model of a microprocessor with l cores can be expressed as ordinary differential equations, such as:

$$\begin{aligned} GT(t) + C\dot{T}(t) &= B_c P(t), \\ Y(t) &= LT(t), \end{aligned} \tag{1}$$

where $T(t) \in \mathbb{R}^n$ is the thermal vector representing temperatures of n blocks of the processor, which includes l cores (with $l < n$) and boundary condition nodes and nodes for package components; $G \in \mathbb{R}^{n\times n}$ includes thermal resistance information; $C \in \mathbb{R}^{n\times n}$ includes thermal capacitance information; $B_c \in \mathbb{R}^{n\times l}$ contains the power injection topology information; $P(t) \in \mathbb{R}^l$ is the power vector with power dissipations of l cores at time k, and is the input of the model; $Y(t)$ is the thermal vector with temperature information of l cores, and is the output of the model; and $L \in \mathbb{R}^{l\times n}$ is the output selection matrix, which selects the l core temperatures from $T(t)$.

In order to analyze the thermal system, the original ordinary differential equation (Equation (1)) in continuous time is discretized into the following difference equation by using Eular's method or other numerical integration methods as

$$\begin{aligned} T(k+1) &= AT(k) + B_d P(k), \\ Y(k) &= LT(k), \end{aligned} \tag{2}$$

where the variables $T(k)$, $P(k)$, and $Y(k)$ are the discretized versions of $T(t)$, $P(t)$, and $Y(t)$ in Equation (1), respectively, and A and B_d are formulated using G, C, and B_c, according to the specific numerical integration method used to discretize Equation (1).

For general purposes, the thermal model in Equation (2) is used to compute the core temperatures[1] of the chip ($Y(k)$ as the output) by feeding in the power consumption of the power unites of the chip ($P(k)$ as the input).

2.2. Desired Power Computing Using Model Predictive Control

Section 2.1 has shown that by using the given power input $P(k)$, the core temperatures of the chip $Y(k)$ can be computed with the thermal model in Equation (2), which is sufficient for thermal simulation and estimation. For DTM problems, computing the desired power with a given temperature is also important because DTM needs to operate on the power side to manage the temperature. Sometimes, the steady-state thermal model, which can be obtained by eliminating the capacitance term from Equation (1), is used to compute the power with the given temperature information, due to its simplicity. However, DTM based on the steady-state thermal model will ignore the current thermal state, which is important when making DTM decisions. It also assumes temperature and power dissipation to be roughly steady, which harms DTM effectiveness. In order to mitigate this problem, the transient thermal model in Equation (1) (or the discretized form in Equation (2)) is used in a feedback control scheme or optimization formulation for better power computation in DTM decision making. Although this method is able to take the current thermal state into consideration and handle transient thermal and power effects, the power it computes still cannot lead to a smooth thermal control. This is mainly because this method lacks the ability to look into the future and thus can only obtain the on-step optimal power for thermal control. In this article, we use the model predictive control (MPC)-based power computation method, which extends the transient thermal model in Equation (2) into the predictive form with the ability to look into the future and compute the *future-aware* desired power for smooth and accurate thermal management.

By using a system model in the form of Equation (2), MPC is able to calculate the input adjustment needed in order to track a user-defined output. In order to maximize the performance of the processer under a thermal constraint, the highest temperature allowed (called *ceiling temperature*) for each core, Y_{max}, is usually provided as the user-defined output to be tracked.[2]

First, we define the difference of the state and input variables as

$$\begin{aligned} \Delta T(k) &= T(k) - T(k-1), \\ \Delta P(k) &= P(k) - P(k-1). \end{aligned} \tag{3}$$

Taking the difference of the adjacent two steps of Equation (2), there is

$$\begin{aligned} \Delta T(k+1) &= A\Delta T(k) + B_d \Delta P(k), \\ Y(k+1) - Y(k) &= LA\Delta T(k) + LB_d \Delta P(k). \end{aligned} \tag{4}$$

Introducing a new variable,

$$\hat{T}(k) = \begin{bmatrix} \Delta T(k) \\ Y(k) \end{bmatrix},$$

[1]Temperatures of other blocks can be computed by simply modifying L. Fine-grained temperature calculation (e.g., temperatures of functional blocks) is also possible by formulating a larger model, which leads to longer computing time.

[2]The temperature ceiling can be adjusted according to real-world applications; it can be also slightly lower than the highest temperature allowed for absolute safety considerations.

we can rewrite Equation (4) into the following augmented model:

$$\begin{aligned} \hat{T}(k+1) &= \hat{A}\hat{T}(k) + \hat{B}\Delta P(k), \\ Y(k) &= \hat{L}\hat{T}(k), \end{aligned} \tag{5}$$

where

$$\hat{A} = \begin{bmatrix} A & 0_m \\ LA & I \end{bmatrix}, \qquad \hat{B} = \begin{bmatrix} B_d \\ LB_d \end{bmatrix},$$

$$\hat{L} = \begin{bmatrix} 0_m & I \end{bmatrix}, \qquad \hat{T}(k) = \begin{bmatrix} \Delta T(k) \\ Y(k) \end{bmatrix},$$

with 0_m as a matrix with all zero elements with suitable size.

So far, we have obtained the connection between the power input difference and the output core temperatures in Equation (5). Next, the power input difference needs to be determined given the desired ceiling temperatures of cores. Assume the ceiling temperatures of cores over several time steps into the future are given, and written in a vector form as

$$Y_{ceil} = \left[Y_{max}^T, Y_{max}^T, \ldots, Y_{max}^T\right]^T \in \mathbb{R}^{mN_p \times 1}.$$

In this vector, $Y_{max} \in \mathbb{R}^{m\times 1}$ contains the ceiling temperatures of each core. Here we assume that the ceiling temperature does not change over time, which usually fits the real-world situation. Please note that this is not a limitation of the new method. N_p stands for a time frame from current to the N_p steps into the future, and is called the prediction horizon. In order to keep the core temperatures tracking the ceiling temperature in the prediction horizon, at a time k, the future control trajectory (which is actually unknown and needs to be computed in the end) is introduced as

$$\Delta P_k = [\Delta P(k), \Delta P(k+1), \ldots, \Delta P(k+N_c-1)]^T,$$

where N_c is called the control horizon. The prediction of core temperatures is defined as

$$Y_k = [Y(k+1|k)^T, Y(k+2|k)^T, \ldots, Y(k+N_p|k)^T]^T,$$

where $Y(k+j|k)$ is the predicted core temperatures at time $k+j$ using information of current time k. Y_k can be calculated assuming ΔP_k is known, using

$$Y_k = V\hat{T}(k) + \Phi\Delta P_k, \tag{6}$$

where V and Φ are shown in Equation (7) as

$$V = \begin{bmatrix} \hat{L}\hat{A} \\ \hat{L}\hat{A}^2 \\ \vdots \\ \hat{L}\hat{A}^{N_p} \end{bmatrix}, \Phi = \begin{bmatrix} \hat{L}\hat{B} & 0 & 0 & \cdots & 0 \\ \hat{L}\hat{A}\hat{B} & \hat{L}\hat{B} & 0 & \cdots & 0 \\ \hat{L}\hat{A}^2\hat{B} & \hat{L}\hat{A}\hat{B} & \hat{L}\hat{B} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \hat{L}\hat{A}^{N_p-1}\hat{B} & \hat{L}\hat{A}^{N_p-2}\hat{B} & \hat{L}\hat{A}^{N_p-3}\hat{B} & \cdots & \hat{L}\hat{A}^{N_p-N_c}\hat{B} \end{bmatrix}. \tag{7}$$

Next, we would like to calculate the power, which minimizes the difference between core temperatures Y_k generated by such power and the desired ceiling temperatures Y_{ceil} given by the user. We can first introduce the measurement of such difference as $(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$, and the optimal power distribution is the one leading to $Y_k = Y_{ceil}$. In addition, for practical considerations, we prefer the power distribution not to change drastically. So the extra tuning term $\Delta P_k^T R \Delta P_k$ is added to $(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$, which forms

$$F = (Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k) + \Delta P_k^T R \Delta P_k \tag{8}$$

as the final function to be minimized over the variable ΔP_k. $R = rI_{N_c \times N_c}$ is the tuning matrix with r as the tuning parameter, which determines the weight between the two terms, and can be fine-tuned through experiment according to different numbers of cores. Guidance and examples of how to tune r are provided in Section 4. Also note that Y_k is a function of the unknown variable ΔP_k.

Next, optimization is performed to minimize Equation (8) by taking the first derivative of Equation (8) with respect to ΔP_k and making it equal to zero. The solution of ΔP_k is

$$\Delta P_k = (\Phi^T \Phi + R)^{-1} \Phi^T (Y_{ceil} - V\hat{T}(k)). \tag{9}$$

At each MPC time k, we only use the first computed control signal $\Delta P(k)$ from Equation (9) and update the power distribution as

$$\bar{P}(k) \leftarrow P(k) + \Delta P(k), \tag{10}$$

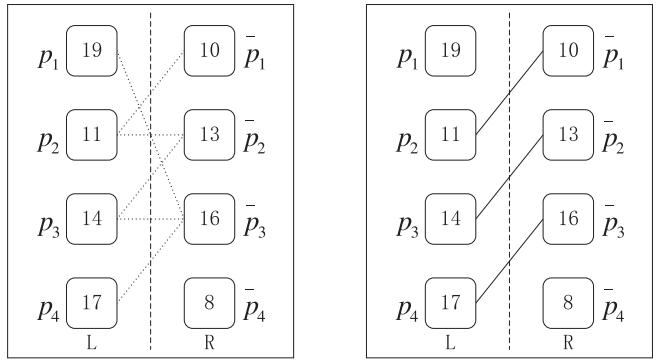
where $\bar{P}(k)$ is the updated power distribution. The resulting temperature $Y(k)$ would track the desired ceiling temperature with the updated power input. In other words, the updated power input is the highest power that can be reached without violating the temperature requirements.

2.3. Task Migration and DVFS Based on Desired Power

Next, dynamic thermal management can be performed with the desired power distribution of cores provided by MPC in Equation (10). DVFS can be integrated with MPC easily by adjusting the frequency and voltage level of each core to match the desired power distribution from MPC. However, DVFS may lead to dramatic performance degradation of thhe microprocessor. The basic reason is that DVFS can only lower the power of the core, but it is not able to increase the power if the core is already at its highest frequency and voltage level, for example, if the load currently running at core i consumes power p_i with the highest voltage level and frequency speed, while MPC suggests this core consumes power $\bar{p}_i$ with $\bar{p}_i > p_i$. In this case, nothing can be done by DVFS at the ith core. But at the same time, there may exist a jth core consuming $p_j \approx \bar{p}_i$, which must be scaled to a lower power (e.g., equal to p_i) by DVFS in order to satisfy the thermal constraint. This leads to performance degradation of the jth core and will be revealed as lower throughput.

Actually, it is obvious that if we simply swap the loads at the jth core and the ith core in this example, no DVFS will be executed and the performance of the microprocessor will not be harmed. As a result, task migration can be performed first by matching similar valued elements in P and $\bar{P}$ into pairs and making the migration action according to the matched pairs. This matching process is an assignment problem, and it can be modeled as a bipartite graph and solved as a bipartite matching problem. The corresponding bipartite graph can be formulated as $\mathcal{G} = (L, R, E)$, where $L = \{p_1, p_2, \ldots, p_l\}$, $R = \{\bar{p}_1, \bar{p}_2, \ldots, \bar{p}_l\}$, and E contains *partial* edges between vertices in L and R: we define a threshold e_{th}, and only the edges $(p_i, \bar{p}_j)$ satisfying $|p_i - \bar{p}_j| < e_{th}$ are kept in E with weights $e_{ij} = |p_i - \bar{p}_j|$. The bipartite matching problem can be solved using the Hungarian algorithm, and the matched pairs are found, which means task migration can be performed according to the pairs (if $(p_i, \bar{p}_j)$ is one of the matched pairs, then the load at the ith core will be moved to the jth core). There may exist some unmatched powers left after the bipartite match, which will be processed by DVFS as introduced later.

The value of e_{th} determines the number of unmatched powers after bipartite match and also controls the risk of temperature violation: a larger value of e_{th} results in fewer unmatched powers, fewer DVFS actions, and higher risk and severity of thermal violation, but higher performance of the chip. The proper value of e_{th} should be determined



(a) The weighted bipartite graph with the threshold $e_{th} = 3$.

(b) The result of the bipartite matching. The matched pairs are connected by solid lines.

Fig. 1. An example showing the weighted bipartite matching.

as the one leading to acceptable thermal violation risk and severity. e_{th} needs to be changed for different CPU architectures with different numbers of cores, but it is not necessarily to be changed at runtime for different workloads. Details about how to adjust e_{th} are discussed in Section 4.

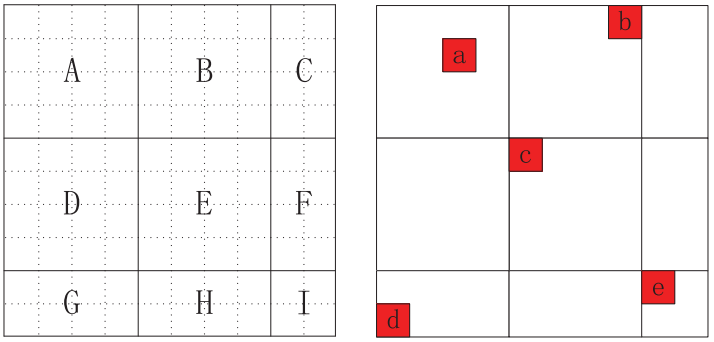
Another important function of e_{th} is to eliminate unnecessary task migrations when there are many low-power tasks. Consider the extreme situation that all tasks are low power tasks, which will result in low temperatures for all cores. In such case, we should not perform any task migration and DVFS. If we set the proper e_{th}, then no task migration will be performed (which is correct), as the bipartite graph has no edge at all. This is because elements in P all have small values, while elements in $\bar{P}$ all have large values (as suggested by MPC in order to track the ceiling temperature), so e_{th} will prevent them from connecting to each other, thus avoiding unnecessary task migrations.

One example of the bipartite matching is shown in Figure 1, with Figure 1(a) showing the weighted bipartite graph with threshold $e_{th} = 3$, and Figure 1(b) demonstrating the matched pairs $(p_2, \bar{p}_1)$, $(p_3, \bar{p}_2)$, and $(p_4, \bar{p}_3)$.

3. HIERARCHICAL DYNAMIC THERMAL MANAGEMENT METHOD

In this section, a new hierarchical DTM method is proposed for high-performance many-core microprocessors. The new technique is based on model predictive control and uses both task migration and DVFS.

It is challenging to perform MPC-based task migration on many-core microprocessors, because when the number of cores becomes large, a lot of time is spent in computing the migration decision making (can be formulated as bipartite matching) using the Hungarian algorithm with complexity of $O(n^3)$. In order to handle the many-core system that has a large core number, the new technique clusters the cores into blocks and makes task migration decision on two levels: within block (lower level) and among blocks (higher level). Bipartite matching of current powers and desired powers is first performed at the lower level inside each block. There are unmatched powers from each block in the lower level, and they are collected to form the upper level (among blocks). At the upper level, an iterative minimum cut algorithm modified from Fidducia and Mattheyses [1982] is used to divide the upper level into "optimal" blocks, and bipartite matching is performed inside each "optimal" block. The final unmatched powers from



(a) Adjacent cores are clustered into blocks preparing for the lower level bipartite matching.

(b) The cores (in red) remain unmatched after the lower level bipartite matching.

Fig. 2. Vertical view of the 100-core microprocessor used as an example to show the hierarchical algorithm.

the upper level are processed using the DVFS method to guarantee the absolute safety in temperature. The hierarchical algorithm relaxes the computational cost by reducing the size of each bipartite matching and performing the matching in parallel. As a result, it is scalable to many-core systems.

3.1. Lower-Level Task Migration Within Blocks

First, we cluster the cores into blocks. As the first step, we can simply cluster the cores according to their locations; that is, we simply group the spatially clustered cores into a block. This lower-level clustering process introduces no overhead at all. We usually form blocks in square shapes called *regular blocks*, but rectangular or smaller square-shaped blocks may appear on the edges, and they are called *edge blocks*.

Bipartite matching, shown in Section 2.3, is performed inside each block, and this is called *lower-level matching*. For each block, computation of the matching can be assigned to a core inside the block. In the view of the full chip, the lower-level matching is performed in parallel. As a result, the latency introduced by lower-level matching is the CPU time used to compute the matching in a regular block (note that all edge blocks have a smaller size than the regular block, which means their shorter computing time will not be counted for latency).

The number of cores inside each block can be tuned to achieve a smaller latency of the whole algorithm: if a large number is chosen here, bipartite matching in the lower level will take more time, but more matched pairs will be found. This will result in fewer unmatched pairs to be left to the upper level, causing less processing time at the upper level.

An example of the lower-level clustering process is shown in Figure 2(a). It is a 100-core microprocessor, and the cores are clustered into four 16-core regular blocks (labeled as A, B, D, E). Five edge blocks (labeled as C, F, G, H, I) are also introduced with the number of cores ranging from four to eight. Lower-level bipartite matching is then performed in each block, with Figure 1 showing the bipartite matching inside block I of Figure 2(a).

Obviously, it is insufficient to find all matching pairs by only performing the lower-level matching inside each block. For example, in block I of Figure 2(a), the powers p_1 and $\bar{p}_4$ cannot be matched as shown in Figure 1(b). It is also not good to perform DVFS for unmatched powers in each block at this lower-level stage, because an unmatched

power in one block may find a good match with another unmatched power from a different block. This will avoid a lot of unnecessary DVFS actions and minimize the performance degradation of the chip. As a result, we can collect all the unmatched powers from all blocks to form the upper level. All the unmatched powers after the first-level matching in the 100-core example are shown in Figure 2(b), marked in red.

3.2. Higher-Level Task Migration Among Blocks

In the previous subsection, we have clustered the cores into blocks and done bipartite matching at the lower level inside each block. All the unmatched powers from all blocks are collected for the upper-level matching.

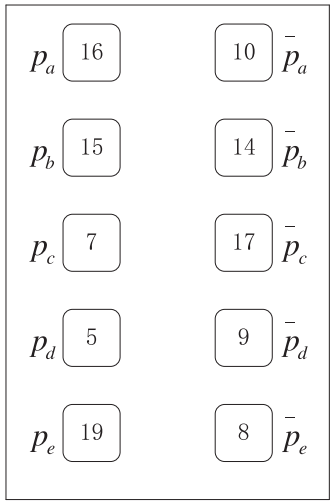
We want to find all the power pairs that can be matched at the upper level and perform DVFS only on the final unmatched powers. It appears that we can cluster the upper-level powers into larger blocks according to their locations, similar to what we have done at the lower level. Then we can perform bipartite matching inside the new blocks and form even larger blocks using previously unmatched ones. The area of the block grows in iteration, until the full chip finally becomes one block. However, experiments show that only a small ratio (below 25%) of powers at the higher level can be finally matched, compared with a large ratio (above 60%) of that at the lower level. Such a small matching ratio will cause the block area to increase very slowly in iteration. It is even possible that the block area will never grow to the size of the full chip because there may be a lot of powers that cannot be eventually matched. So before the block enlarges to the chip size, unmatched powers collected in the block may already be too many to be processed, due to the large computing cost required.

In order to make the task migration decision efficiently at a higher level, we cluster the upper-level powers into blocks in another way, using the minimum cut algorithm. First, we formulate a graph (details will be provided later) using all the upper-level powers. Then, we use minimum cut algorithm to divide the graph into two groups, and each group is a new block.[3] If the new block size is too large (for the bipartite matching algorithm), then another minimum cut is performed on this block to half its size. After the minimum cut, bipartite matching is performed inside each new block. One important property of the minimum cut is that the connection is weak among the separated blocks, and the connection is much stronger inside each block. This means the matching ratio is maximized inside each block. If there are unmatched powers left after the bipartite matching inside a block, these unmatched powers can also hardly be matched with powers from other blocks. As a result, another round of even higher-level matching by collecting all unmatched powers from all upper-level blocks is unnecessary.

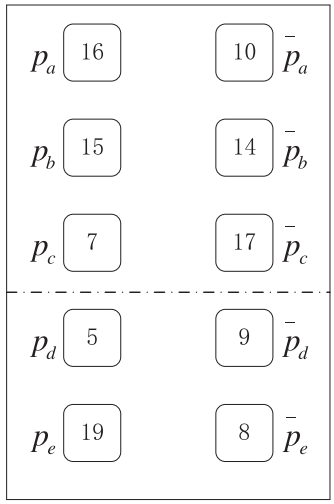
Usually, the exact minimum cut algorithm is too expensive to be performed here as a runtime algorithm. Fortunately, we do not need the optimal cut here, and we can use the iterative approximation algorithm modified from the one used in network partition [Fidducia and Mattheyses 1982]. First, we build a new graph $\mathcal{G}_p = (V_p, E_p)$ using all the upper-level powers, where $V_p = \{p_1, \bar{p}_1, p_2, \bar{p}_2, \ldots, p_m, \bar{p}_m\}$, and E_p contains all connections of vertices in V_p with weights. The weights are defined as $w(p_i, \bar{p}_j) = 1/|p_i - \bar{p}_j|$, $w(p_i, p_j) = 0$, and $w(\bar{p}_i, \bar{p}_j) = 0$ for all the i and j. One example of $\mathcal{G}_p$ is shown in Figure 3(a). What we need to solve here is a minimum cut problem on the graph $\mathcal{G}_p$.

As an iterative algorithm, the first step is to make the initial cut by partitioning V_p into two subsets V_{p1} and V_{p2}. The *cost* of the cut is defined as the sum of the weights on the cut set. An example is given in Figure 3(b), where the initial cut generates two subsets $V_{p1} = \{P_a, P_b, P_c, \bar{P}_a, \bar{P}_b, \bar{P}_c\}$ and $V_{p2} = \{P_d, P_e, \bar{P}_d, \bar{P}_e\}$.

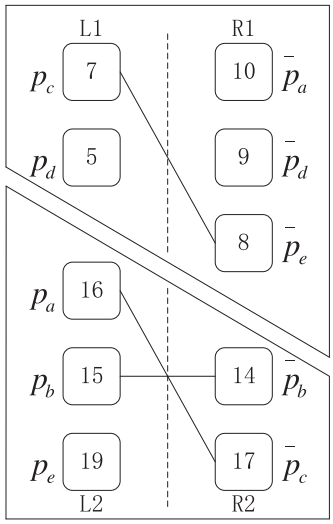
[3]Cores in this upper-level block are unnecessarily spatially adjacent to each other.



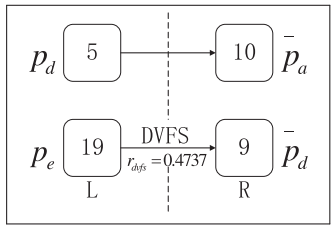
(a) Graph $\mathcal{G}_p$ (edges and weights are not shown for simplicity).



(b) An initial cut is performed on $\mathcal{G}_p$ as the first step of the modified iterative minimum cut algorithm.



(c) A minimum cut is performed on $\mathcal{G}_p$, followed with the bipartite matching in each upper level block.



(d) Final adjustment by DVFS on the unmatched powers.

Fig. 3. The higher-level processing example of the hierarchical algorithm on the 100-core microprocessor.

Then, in order to proceed with the iteration by moving the correct vertices from one subset to the other one, we need to measure the cut cost changes of a moving action. For each vertex v, we first define $I(v)$ to be the set of edges that connect v and another vertex in the *same* subset as v. Similarly, we define $E(v)$ to be the set of edges that connect v and another vertex in the *different* subset as v. Take vertex P_a in Figure 3(b) as an example; $I(P_a) = \{(P_a, \bar{P}_a), (P_a, \bar{P}_b), (P_a, \bar{P}_c)\}$, and $E(P_a) = \{(P_a, \bar{P}_d), (P_a, \bar{P}_e)\}$. Note that here we ignore the edges with 0 weight, such as (P_a, P_b). The following *gain* function $f(v)$:

$$f(v) = \sum_{n_i \in E(v)} w(n_i) - \sum_{n_j \in I(v)} w(n_j) \tag{11}$$

measures the *decrease* in cut cost if v is moved to the other subset. In the example, $f(P_a)$ is calculated as $f(P_a) = (1/|P_a - \bar{P}_d| + 1/|P_a - \bar{P}_e|) - (1/|P_a - \bar{P}_a| + 1/|P_a - \bar{P}_b| + 1/|P_a - \bar{P}_c|) = -1.40$. The negative value of $f(P_a)$ means that the action of moving P_a to the other subset is going to increase the cut cost, which indicates that P_a should not be moved. Similarly, the gains of other vertices are calculated as $f(P_b) = -1.39$, $f(P_c) = 0.92$, $f(P_d) = -0.19$, $f(P_e) = 0.62$, $f(\bar{P}_a) = -0.39$, $f(\bar{P}_b) = -1.33$, $f(\bar{P}_c) = -1.02$, $f(\bar{P}_d) = 0.46$, and $f(\bar{P}_e) = 0.84$. For the ith iteration, the best vertex (with the largest gain) is chosen to be moved to the other subset, and this vertex is denoted as v_i. It will be locked in that subset, and the gain of all its neighbors will be updated. In the example, the best vertex is P_c, which has the largest gain of 0.92, and it will be moved to the lower subset and locked. In this simple example, it can be easily verified from the figure that moving P_c from the upper subset to the lower subset increases the power matching quality: P_c is a better matching candidate for both $\bar{P}_d$ and $\bar{P}_e$ in the lower subset than any other power vertices in the upper subset. However, there is one problem if we directly perform this moving decision. In our case, using the best vertex in every iteration may result in an extremely unbalanced result: one subset contains a lot of vertices (powers), while the other subset contains very little. This will cause a problem for our method because if the size of one block after a minimum cut remains large, performing bipartite matching for this block will dominate the latency. As a result, we introduce a *balance threshold* to the iterative minimum cut algorithm: if moving the best vertex from subset A to B violates the threshold, then it will not be moved, and instead, the best vertex in subset B will be moved to A and locked. After all nodes are locked, there is a sequence $\mathcal{F} = \{f(v_1), f(v_2), \ldots, f(v_{2m})\}$. At this stage, it seems strange that we have "moved" all vertices to their corresponding other subsets, which does not make any sense. The reality is that all the moving action now is for the purpose of analysis only, in order to determine which vertices will be *actually moved*, as shown in the following step.

Note that in the sequence $\mathcal{F}$, $f(v_i)$ means the gain (decrease in cut cost) of the single moving action of v_i, assuming all previous moving actions of $v_1, v_2, \ldots, v_{i-1}$ have already been done. As a result, in order to know the total gain of moving $v_1, v_2, \ldots, v_i$, denoted as $\tilde{f}(v_i)$, we need to take the summation as

$$\tilde{f}(v_i) = \sum_{j=1}^{i} f(v_j). \tag{12}$$

By calculating $\tilde{f}(v_i)$ for $i = 1, 2, \ldots, 2m$, we can form the cumulative sum sequence $\tilde{\mathcal{F}} = \{\tilde{f}(v_1), \tilde{f}(v_2), \ldots, \tilde{f}(v_{2m})\}$, where $\tilde{f}(v_i)$ is the total gain of moving vertices from v_1 till v_i as discussed before. Assume $\tilde{f}(v_k)$ is the largest element in $\tilde{F}$; it means that moving $v_1, v_2, \ldots, v_k$ leads to the largest total gain (i.e., largest cut cost decrease). As a

result, we can perform the *actual action* by moving $\{v_1, v_2, \ldots, v_k\}$ to their corresponding other subsets. All aforementioned procedures are counted as *one moving action*.

Although the moving action can be performed repeatedly, previous research on circuit partitioning shows that two to four moving actions are enough to achieve the local minimum [Fidducia and Mattheyses 1982; Dutt and Deng 1996]. For our case, experiments show that only one moving action performs well enough.

After the minimum cuts, we cluster the powers into blocks. Then, bipartite matching can be performed inside each block. Since minimum cuts already grouped the correlated powers into one block, the remaining unmatched powers from one block can hardly be matched with unmatched powers from other blocks. As a result, no further bipartite matching will be performed and we can immediately determine the task migration action based on the previous lower-level and higher-level bipartite matching results. And the final remaining unmatched powers can be simply processed using DVFS.

The example showing the higher-level matching is presented in Figures 3(a), 3(b), and 3(c). In Figure 3(a), all the unmatched powers after the lower-level matching (which can be seen in Figure 2(b) in red) are collected to form the graph $\mathcal{G}_p$. Note that *all* vertices are connected by edges in graph $\mathcal{G}_p$ with weights, but they are not shown in Figure 3 only for simplicity reasons. Then, the iterative minimum cut algorithm is performed on graph $\mathcal{G}_p$, with the initial cut in the dashed line shown in Figure 3(b), and the final cut shown in Figure 3(c) dividing the powers into two blocks. Next, upper-level bipartite matching is performed by formulating a new bipartite graph for each block as shown in Figure 3(c).

3.3. Final Adjustment by DVFS

The DVFS method is introduced to do the final adjustment on the unmatched powers from the previous bipartite matching algorithm. After the task migration action based on the lower- and higher-level bipartite matching results, the already-matched powers have been moved to the correct cores, which matches the power distribution given by MPC with little difference at these positions. Because of such moving action, even the remaining unmatched powers have been moved to a new position since their original positions may have been taken by the matched ones.

Assume a load with unmatched power p_i has been moved to the jth core where the required power given by MPC is $\bar{p}_j$. Since p_i is not matched with $\bar{p}_j$, they cannot equal in value. If $p_i < \bar{p}_j$, it means that if we keep the current state, the resulting temperature around the jth core will be lower than the required ceiling temperature. Since this will not harm the reliability of the chip, we can keep this power unchanged. In the other case where $p_i > \bar{p}_j$, we perform DVFS on p_i with the power scaling ratio $r_{dvfs} = \bar{p}_j / p_i$. This is the maximum performance that can be achieved by the jth core without violating the temperature constraint. It is also noted that DVFS makes discrete voltage/frequency adjustments, so in real-world applications, r_{dvfs} is determined as the nearest level, which is lower than $\bar{p}_j / p_i$.

The final step performed on the 100-core example is shown in Figure 3(d). Since there is $p_d < \bar{p}_a$, p_d is simply moved to its new position without DVFS. And DVFS is performed on p_e because $p_e > \bar{p}_d$.

It is well known that in the DVFS method, the DC-DC converter, which is used to adjust the supply voltage level, introduces overhead in the chip design. This could be a serious problem in many-core systems, especially for per-core DVFS. It is practical to introduce *DVFS block*, which contains a number of spatially adjacent cores that share one DC-DC converter, in order to reduce the implementation overhead of the DC-DC converter. In such case, the DVFS decision of each DVFS block should be made according to the lowest voltage level requested by cores in the corresponding block and leading to a throughput/performance drop. This is a tradeoff between DC-DC converter

implementation overhead against the chip performance. This tradeoff is a general and important problem in many-core architecture and needs to be further researched in future works.

This concludes the new algorithm, and we summarize the whole flow of the new method in Algorithm 1.

ALGORITHM 1: Hierarchical Dynamic Thermal Management Algorithm

1: Calculate the desired power distribution $\bar{P}(k)$ using MPC with the provided temperature constraint as in Equations (9) and (10).
2: Cluster the adjacent cores into lower-level blocks.
3: For each block, build its own bipartite graph $\mathcal{G} = (L, R, E)$ using the corresponding part of $P(k)$, $\bar{P}(k)$, and threshold e_{th}.
4: Perform lower-level bipartite matching for each block. Record the matched pairs.
5: Collect unmatched powers from all lower-level blocks and build a new graph $\mathcal{G}_p = (V_p, E_p)$ for minimum cut.
6: Generate higher-level blocks by partitioning the graph $\mathcal{G}_p$ using the modified iterative minimum cutalgorithm.
7: Perform upper-level bipartite matching for each block generated in Step 6. Record the matched pairs.
8: Determine the new positions of all loads based on the recorded lower-level and upper-level matched pairs.
9: For all remaining unmatched powers, perform DVFS as described in Section 3.3.

4. EXPERIMENTAL RESULTS

The experiments are performed on a Linux server with two 2.90GHz eight-core 16-thread CPUs and 64GB memory. The new hierarchical method is implemented using MATLAB, and HotSpot [Huang et al. 2006] is used to build the thermal model based on the many-core microprocessors with four different core configurations, from 100 cores (10×10) to 625 cores (25×25). The ambient temperature is 20°C. The many-core microprocessors in the experiments are composed of identical Alpha 21264 cores. The dimension of all chips is $10mm \times 10mm \times 0.15mm$. We assume there is no task-processing-related communication and synchronization among cores in the many-core microprocessors; that is, one task is assigned to one core. Wattch [Brooks et al. 2000] is used to generate the power by running SPEC benchmarks [Henning 2000], and one task is assigned to one core randomly as an initial task assignment. Next, task assignment and scheduling actions are determined by the DTM method. We use nine power traces of nine different SPEC benchmarks. For the power traces of different CPUs, we recycle those nine power traces to get 100 power traces, 256 power traces, and so on.

Because the size of the core scales as the core number increases, this will lead to unrealistically high power density (and extremely high temperature) after the so-called "power wall" or "utilization wall" is reached [Taylor 2013]. There are many solutions proposed in order to solve this problem. One popular solution is dim silicon, which scales power consumption of each core [Huang et al. 2011; Taylor 2013], and another well-known solution is turning off some cores entirely [Taylor 2013; Shafique et al. 2014]. In this work, we adopt the dim silicon technology and scale the power traces to ensure that all CPUs have similar power density, which leads to a temperature distribution similar to today's multicore chips. It is achieved by scaling the operating frequency and voltage by the same ratio. In this work, we do not consider the strategy of turning off a portion of cores entirely, which will be our future research direction.

Table I. Parameters of the New Hierarchical Method for CPUs with Different Core Configurations

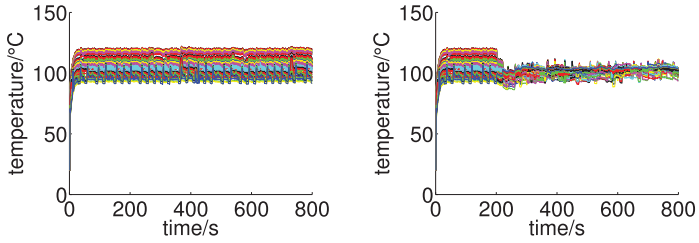
Configuration	Scale	e_{th}	r
100 cores (10×10)	0.21	0.06	500
256 cores (16×16)	0.08	0.05	2,100
400 cores (20×20)	0.052	0.04	3,000
625 cores (25×25)	0.033	0.03	3,000

For CPUs with different core numbers, the threshold e_{th} in the task migration process and the tuning parameter r in MPC (in Equation (8)) are manually tuned for each CPU to ensure the temperature tracking quality. Note that the optimal values of e_{th} and r highly depend on the core number and architecture of the microprocessor. It is hard and unnecessary to calculate these parameters theoretically, and in real-world applications, it is much easier to perform experiments and fine-tune these parameters for a certain many-core microprocessor, even in a trial-and-error way. All these parameters are shown in Table I. We can see that e_{th} scales with the size of the power module (size of the core in our case). If the size of the core is relatively large, e_{th} needs to be assigned a relatively large value as well (please refer to the 100-core case in Table I shown later in the experiment), and vice versa (see the 625-core case in Table I). This is because a larger-power module means a larger area for power dissipation (this power module has a larger power value as well), and as a result is able to allow a larger power value difference for the same temperature violation tolerance. Another observation is that larger number of cores needs a larger r. This is simply because in Equation (8), when the core number increases, the magnitude of each element in $Y_{ceil} - Y_k$ does not change much, but each element in ΔP_k (which is unknown and needs to be solved) should be smaller (because the power of each core decreases). In order to achieve a smaller solution of ΔP_k, a larger r needs to be used. Please note that for 625 cores, r remains to be 3,000, which is the same as for 400 cores, because we found the difference is insignificant by increasing r further.

For the lower-level clustering, we cluster every 25 (5×5) adjacent cores into one block. For the upper-level processing, graph $\mathcal{G}_p$ will be partitioned by the modified iterative minimum cut algorithm if the number of vertices in graph $\mathcal{G}_p$ is more than 240.

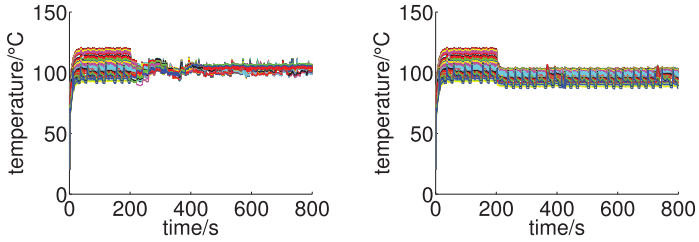
The DVFS and task migration activating period is set to be every 20s to minimize the overhead effect from task migration (overhead from both computing and migration action) when the core number increases. It is usually fine to use a much smaller migration period with a small number of cores, where computing overhead and migration action overhead (due to core-to-core communication) are both small. For the many-core case, frequent task migration is extremely hard to perform considering the large number of cores, so the migration period is extended. It is possible that the load on one core is raised a lot in between the migration intervals and this may cause a temperature violation. In such case, we can just perform DVFS inside the migration intervals when needed to enhance the safety.

The overhead of the task migration action is considered in the experiments in addition to the task migration decision computing overhead. The normal task migration action overhead is around 10^6 cycles, which counts for around 1ms for a 1GHz processor [Cuesta et al. 2010]. Such overhead can be higher in many-core systems, because of the long communication time caused by the large number of cores. As a result, we set the migration time to be 100ms in a system with hundreds of cores. We also consider the power consumption of task migration in the experiments. Such power is mainly caused by the communication between the cores during migration time, and the corresponding two cores are not processing any tasks during migration. As a result, we assume the



(a) Temperature traces without any DTM method.

(b) Temperature traces with the new hierarchical DTM method activated at the 200th second.



(c) Temperature traces with DTM method in Ma et al. [2014] activated at the 200th second.

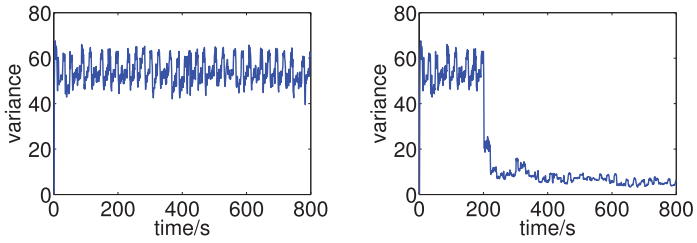
(d) Temperature traces with the DTM method with DVFS only activated at the 200th second.

Fig. 4. Transient temperature traces of the 100-core CPU with different DTM methods. Lines with different colors represent temperature traces of different cores. The new method in (b) [Ma et al. 2014] and in (c) successfully tracks the ceiling temperature. But the DVFS-only method in (d) has many low-temperature cores, which means the chip performance is lower.

power during the task migration time to be lower than the task processing time. In order to be safe, we *pessimistically* make the power of a core during task migration time to be the previous task average power processed on the core, and feed such power into MPC for prediction.

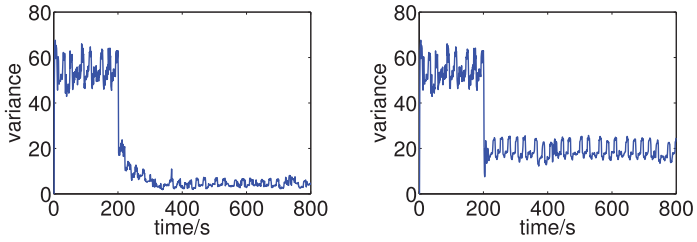
For comparison, we have implemented two other MPC-based methods, the MPC-based method with DVFS and task migration [Ma et al. 2014] for a multicore microprocessor and the MPC-based method with DVFS only. We also choose the DTM method in Hanumaiah et al. [2011] for comparison because it shares the same goal of our work: maximize performance (throughput) with temperature as the constraint in high-performance processors. We have used the open-source program MAGMA V2 provided by Hanumaiah et al. [2011] in the experiment. MAGMA can only give the results of the 100-core case and will fail with "out of memory" error for larger cases. In order to be fair, all methods share the same activation period, power traces, and ceiling temperature settings.

First, we let a CPU with 100 cores run at maximum speed without any DTM method. The transient temperature traces of this CPU are shown in Figure 4(a). We can see that the temperature of the core ranges from 90°C to 120°C. The temperature around 120°C will clearly harm the reliability of the chip. The temperature variance among cores is also measured and shown in Figure 5(a); it can be seen that the temperature



(a) The temperature variance without any DTM method.

(b) The temperature variance with the new hierarchical DTM method activated at the 200th second.



(c) The temperature variance with DTM method in Ma et al. [2014] activated at the 200th second.

(d) The temperature variance with the DTM method with DVFS-only activated at the 200th second.

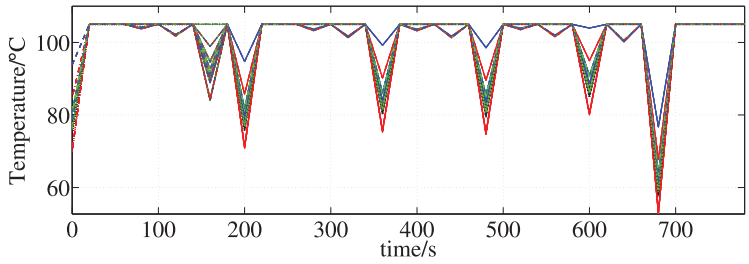
Fig. 5. Transient variances among cores in the 100-core CPU with different DTM methods. The temperature variance of the new method is slightly higher than that of the method in Ma et al. [2014], but much lower than that of the DVFS-only method.

difference among cores is large without any DTM method activated.

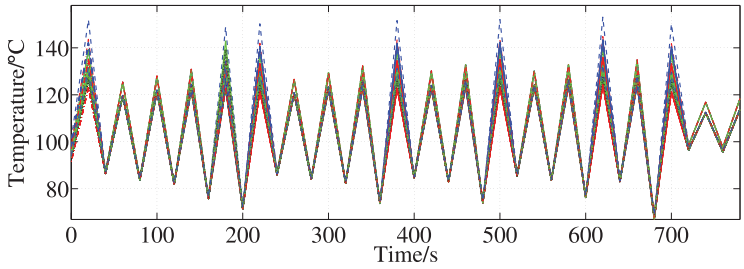
Next, we test the new hierarchical method with the ceiling temperature set as 105°C. The new hierarchical method is activated at the time of 200 seconds. The corresponding transient temperature traces of the 100-core CPU are shown in Figure 4(b), and the corresponding temperature variance is shown in Figure 5(b). After the new hierarchical method is activated, the temperatures of all cores track the given 105°C ceiling temperature, and the temperature difference is greatly decreased.

For comparison, the MPC-based method with task migration and DVFS for multicore microprocessors in Ma et al. [2014] and the MPC-based method with DVFS only are tested first. The ceiling temperatures of all methods are set to be 105°C, and all methods start to be activated at the same time point of 200 seconds with an activating period of 20s. Figures 4(c) and 4(d) and 5(c) and 5(d) show the corresponding transient temperature traces and temperature variances. The new hierarchical method has similar transient behavior and a slightly higher variance compared with the method in Ma et al. [2014] (but the new method performs way better in overhead and scalability as will be shown later). And the DVFS-only MPC-based method performs much worse than both the new hierarchical method and Ma et al. [2014] in terms of transient temperature control and variance among cores.

Next, we compare our results with the method in Hanumaiah et al. [2011]. It also performs DVFS and task migration in order to keep the temperature of the cores tracking the ceiling temperature. However, in Hanumaiah et al. [2011], at making DTM decisions, each core is assumed to be thermally isolated with all other cores in order to save the computing cost. Such assumption may hold for a multicore chip



(a) Temperature traces of cores by assuming no heat exchange among cores.



(b) Actual temperature traces of cores by enabling heat exchange among cores.

Fig. 6. Transient temperature traces of cores by performing DTM in Hanumaiah et al. [2011] on the 100-core microprocessor. Core-to-core heat exchange has a huge impact on temperatures for many-core microprocessors, causing the 105°C ceiling temperature to be seriously violated. The overshoot problem is also significant compared to the MPC-based techniques.

with small core numbers, where the large cache area blocks heat exchange among cores. But in many-core cases, due to the small size of each core, heat exchange among cores is significant and cannot be ignored. Figure 6(a) shows the computed temperatures of the 100-core microprocessor without considering heat exchange among cores. Note that the temperature waveforms are in straight segments because of the larger simulation steps used. Since DTM decisions are made by such temperatures, the ceiling temperature is successfully tracked for most of the time. But such temperatures are not the actual ones because a zero core-to-core heat exchange assumption is made. We modified the MAGMA program and plotted the actual temperatures considering heat exchange among cores. The results are shown in Figure 6(b). It is clear that the DTM decision made with such an assumption is not optimal, and causes significant actual temperature violations. From Figure 6, we can also see the overshoot problem in temperature control. It is because of the zero-slack policy used for temperature/power control in Hanumaiah et al. [2011]: when the current temperature is not exactly the same as the ceiling temperature, the core is either turned off (if current temperature is higher than the ceiling temperature) or run at full speed (if current temperature is lower than the ceiling temperature). All the MPC-based algorithms do not have such an overshoot problem as shown in Figure 4, because they are able to predict into the future and make DTM decisions, which leads to smooth temperature control.

The variance comparisons on all CPUs with different numbers of cores are recorded in Table II, where var_o is variance without any DTM method, var_d is variance with DTM using DVFS only, var_c is variance with DTM in Ma et al. [2014], var_n is variance with our new hierarchical method, and var_h is variance with the method in Hanumaiah et al. [2011]. All results for different numbers of cores are similar to the 100-core example:

Table II. The Temperature Variance Among Cores in CPUs with Different Core Configurations

Configuration	var_o	var_d	var_c	var_n	var_h
100 cores (10×10)	54.2	18.8	5.5	7.6	5.9
256 cores (16×16)	50.8	19.3	3.4	6.5	N/A
400 cores (20×20)	52.1	16.7	2.5	4.1	N/A
625 cores (25×25)	50.5	15.9	2.3	4.1	N/A

var_o is the variance without any DTM method, var_d is variance with DTM using DVFS only, var_c is variance with DTM in Ma et al. [2014], var_n is variance with our new hierarchical method, and var_h is the variance with method in Hanumaiah et al. [2011].

Table III. Average Runtime Per Second of the DTM Methods on CPUs with Different Core Configurations

	New Method				Ma et al. [2014]			DVFS only		Hanumaiah et al. [2011]	
Core #	t_p ($10^{-4}s$)	t_m ($10^{-4}s$)	t_a ($10^{-4}s$)	t_{all} ($10^{-4}s$)	t_p ($10^{-4}s$)	t_m (s)	t_{all} (s)	t_p ($10^{-4}s$)	t_{all} ($10^{-4}s$)	t_m (s)	t_{all} (s)
100	1.58	1.63	0	3.21	1.49	0.01	0.01	1.60	1.60	0.01	0.01
256	2.85	19.26	1.58	23.7	2.93	0.45	0.45	2.80	2.80	0.09	0.09
400	4.88	7.77	3.27	15.9	5.38	1.90	1.90	5.29	5.29	0.34	0.34
625	9.09	12.27	17.49	38.8	8.27	8.63	8.63	8.87	8.87	0.99	0.99

t_p represents MPC time, t_m means task migration matching time, and t_a denotes for minimum cut partitioning time.

the method in Ma et al. [2014] slightly outperforms the new hierarchical method, while the DVFS-only MPC-based method has significantly larger variance among cores. The method in Hanumaiah et al. [2011], although it can only finish the 100-core case, performs well in the variance test and gives similar results to the method in Ma et al. [2014].

It can be seen from previous comparisons for transient temperature traces and temperature variance that the difference between our new hierarchical method and the method in Ma et al. [2014] is small. What really shines for the new hierarchical method is the scalability and small overhead for many-core microprocessors compared with Ma et al. [2014] and Hanumaiah et al. [2011]. Now, we measure the average runtime-per-second execution for all the DTM methods. Since the new method is mainly composed of MPC, bipartite matching, and minimum cut partitioning, we split the total runtime into MPC time t_p, matching time t_m, and minimum cut partitioning time t_a for better analysis. The runtime of the matching operation of the new method is measured by adding one lower-level matching time and one upper-level matching time. For lower-level (upper-level) matching time, if there are several matching actions performed in parallel, we count the longest one that is dominating the delay. The method in Ma et al. [2014] shares MPC time t_p and t_m but does not have partitioning time t_a. The DVFS-only method shares only MPC time t_p. Although the method in Hanumaiah et al. [2011] can only finish the 100-core case, we are still able to test its task migration time t_m by feeding the corresponding function using a matrix with correct dimensions filled by random numbers. The computing times of CPUs with different numbers of cores are recorded in Table III. It is obvious that for the method in Ma et al. [2014], the overhead becomes more significant as the core number increases. Starting from the 400-core case, for each second of management, this method spends more than 1 second on computing, which makes it totally inapplicable. For the method in Hanumaiah et al. [2011], the task migration computing time also increases quickly with the core number, which makes it unusable for many-core applications. According to their paper, this is because the task migration algorithm suffers the $O(nq)^3$ complexity (which is similar

Table IV. The Average Number of Instructions Per Second of One Core on CPUs with Different Configurations

Configuration	$MIPS_o$	$MIPS_d$	$MIPS_n$
100 cores (10×10)	290.8	279.6	281.5
256 cores (16×16)	210.2	202.9	207.1
400 cores (20×20)	182.1	174.7	178.8
625 cores (25×25)	156.5	150.2	154.4

$MIPS_o$ represents the IPS in millions (MIPS) of the core without any DTM method, $MIPS_d$ is for MIPS of DTM with DVFS only, and $MIPS_n$ denotes MIPS of our new hierarchical method.

to the task migration complexity in Ma et al. [2014]), where n is the core number and q is the task number, and $n = q$ is assumed in Hanumaiah et al. [2011]. When the core/task number increases, such $O(n^6)$ complexity will make the task migration too time consuming to be used. On the other hand, the computing time grows very slowly in the new method. Even for the 625-core case, which is considered as a huge number of cores, the new method is able to make management decisions in 4ms for every 1 second of management. This is only 0.4% of the computing time spent on only a few number of cores, and thus can be considered as negligible. Of course, the DVFS-only method performs best in overhead, but the slight overhead spent for task migration in the new method brings significant advantage in CPU performance, as shown next.

Finally, we measure the performance of different many-core CPUs using different DTM methods. Instructions per second (IPS) is used to estimate CPU performance. Since the methods in Ma et al. [2014] and Hanumaiah et al. [2011] show significant overhead and cannot complete the management decision in time for many-core CPUs, they are not considered in this CPU performance comparison. The average IPS of the core using the new hierarchical method and the method with DVFS only are collected in Table IV. In the table, $MIPS_o$ represents the average number of IPS in millions (MIPS) of the core without any DTM method, $MIPS_d$ stands for the average number of IPS in millions of the core using DTM with DVFS only, and $MIPS_n$ denotes the average number of IPS in millions of the core with our new hierarchical method. Considering $MIPS_o$ as the ideal performance of the CPU without any thermal constraint, $MIPS_n$ is only slightly smaller than $MIPS_o$, showing the effectiveness of the new hierarchical algorithm in optimal management decision making and small overhead even for a huge number of cores. The new method also outperforms the DVFS-only method in terms of CPU performance. Although the new method is slightly larger in overhead, the time spent in task migration decision making reduces the number of DVFS activation times and brings overall performance benefits. We remark that the throughput enhancement of the new method compared to the DVFS-only method is highly dependent on the running applications. If there exists very low-temperature cores (or even idle cores), then the throughput improvement of the new method will be much more significant compared to the DVFS-only method, because the very low-temperature cores can be fully used in the migration process to reduce the DVFS actions.

5. CONCLUSION

In this article, a hierarchical dynamic thermal management method has been proposed for high-performance many-core microprocessors. Based on model predictive control, the new method uses both task migration and DVFS to reduce performance degradation and improve the thermal reliability of the chip. In order to be scalable for many-core microprocessors, it performs bipartite matching-based task migration decision making at two levels: lower level within block and higher level among blocks.

A modified iterative minimum cut algorithm is used to assist the upper-level task migration decision-making process. Experiments on a number of many-core microprocessors show that the new method is able to keep the chip in a safe temperature range, and it outperforms existing methods with higher computing performance of many-core microprocessors.

REFERENCES

Raid Ayoub and Tajana Rosing. 2009. Predict and act: Dynamic thermal management for multi-core processor. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED'09)*. 99–104.

Andrea Bartolini, Matteo Cacciari, Andrea Tilli, and Luca Benini. 2013. Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller. *IEEE Transactions on Parallel and Distributed Systems* 24, 1 (January 2013), 170–183.

David Brooks, Robert Dick, Russ Joseph, and Li Shang. 2007. Power, thermal, and reliability modeling in nanometer-scale microprocessors. *IEEE Micro* 27, 3 (May–June 2007), 49–62.

David Brooks and Margaret Martonosi. 2001. Dynamic thermal management for high-performance microprocessors. In *Proceedings of the IEEE International Symposium on High-Performance Computer Architecture (HPCA'01)*. 171–182.

David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the International Symposium on Computer Architecture (ISCA'00)*. 83–94.

Thidapat Chantem, Sharon Hu, and Robert Dick. 2011. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 10 (October 2011), 1884–1897.

Jason Cong and Bo Yuan. 2012. Energy-efficient scheduling on heterogeneous multi-core architectures. In *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED'12)*. 345–350.

David Cuesta, Jose Ayala, Jose Hidalgo, David Atienza, Andrea Acquaviva, and Enrico Macii. 2010. Adaptive task migration policies for thermal control in MPSoCs. In *Proceedings of the IEEE Annual Symposium on VLSI*. 110–115.

James Donald and Margaret Martonosi. 2006. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the International Symposium on Computer Architecture (ISCA'06)*. 78–88.

Shantanu Dutt and Wenyong Deng. 1996. A probability-based approach to VLSI circuit partitioning. In *Proc. Design Automation Conf. (DAC'96)*. ACM, 100–105.

Thomas Ebi, Mohammad Abdullah Al Faruque, and Jörg Henkel. 2009. TAPE: Thermal-aware agent-based power economy for multi/many-core architectures. In *Proceedings of the International Symposium on-Computer Aided Design (ICCAD'09)*. 302–309.

Charles Fidducia and Robert Mattheyses. 1982. A linear-time heuristic for improving network partitions. In *Proceedings of the Design Automation Conference (DAC'92)*. 175–181.

Yang Ge, Parth Malani, and Qinru Qiu. 2010. Distributed task migration for thermal management in many-core systems. In *Proceedings of the Design Automation Conference (DAC'10)*. 579–584.

Vinay Hanumaiah and Sarma Vrudhula. 2014. Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling. *IEEE Transactions on Computers* 63, 2 (February 2014), 349–360.

Vinay Hanumaiah, Sarma Vrudhula, and Karam Chatha. 2011. Performance optimal online DVFS and task migration techniques for thermally constrained multi-core processors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30, 11 (November 2011), 1677–1690.

John L. Henning. 2000. SPEC CPU 2000: Measuring CPU performance in the new millennium. *IEEE Computer* 1, 7 (July 2000), 28–35.

Wei Huang, Shougata Ghosh, Siva Velusamy, Karthik Sankaranarayanan, Kevin Skadron, and Mircea R. Stan. 2006. HotSpot: A compact thermal modeling methodology for early-stage VLSI design. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 14, 5 (May 2006), 501–513.

Wei Huang, Karthick Rajamani, Mircea Stan, and Kevin Skadron. 2011. Scaling with design constraints: Predicting the future of big chips. *IEEE Micro* 31, 4 (July–August 2011), 16–29.

Ramkumar Jayaseelan and Tulika Mitra. 2009. A hybrid local-global approach for multi-core thermal management. In *Proceedings of the International Symposium on Computer Aided Design (ICCAD'09)*. 314–320.

Michael Kadin, Sherief Reda, and Augustus Uht. 2009. Central versus distributed dynamic thermal management for multi-core processors: Which one is better? In *Proceedings of the IEEE/ACM International Great Lakes Symposium on VLSI (GLSVLSI'09)*. 137–140.

Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jorg Henkel. 2015. Thermal constrained resource management for mixed ILP-TLP workloads in dark silicon chips. In *Proceedings of the Design Automation Conference (DAC'15)*. 1–6.

Guanglei Liu, Ming Fan, and Gang Quan. 2012. Neighbor-aware dynamic thermal management for multi-core platform. In *Proceedings of the European Design and Test Conference (DATE'12)*. 187–192.

Jian Ma, Hai Wang, Sheldon Tan, Chi Zhang, and He Tang. 2014. Hybrid dynamic thermal management method with model predictive control. In *Proceedings of the IEEE Asia Pacific Conference on Circuits and Systems*.

Almir Mutapcic, Stephen Boyd, Srinivasan Murali, David Atienza, Giovanni De Micheli, and Rajesh Gupta. 2009. Processor speed control with thermal constraints. *IEEE Transactions on Circuits and Systems I: Regular Papers* 56, 9 (September 2009), 1994–2007.

Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. 2013. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proceedings of the Design Automation Conference (DAC'13)*. 1–9.

Michael Powell, Mohamed Gomaa, and T. N. Vijaykumar. 2004. Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'04)*. 260–270.

Muhammad Shafique, Shelly Garg, Jorg Henkel, and Dinan Marculescu. 2014. The EDA challenges in the dark silicon era. In *Proc. Design Automation Conf. (DAC'14)*. 1–6.

Kevin Skadron, Mircea Stan, Wei Huang, Siva Velusamy, Karthik Sankaranarayanan, and David Tarjan. 2003. Temperature-aware microarchitecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA'03)*. 2–13.

Cheng Tan, Thannirmalai Muthukaruppan, Tulika Mitra, and Lei Ju. 2015. Approximation-aware scheduling on heterogeneous multi-core architectures. In *Proceedings of the Asia South Pacific Design Automation Conference (ASP-DAC'15)*. 618–623.

Michael Taylor. 2013. A landscape of the new dark silicon design regime. *IEEE Micro* 33, 5 (October 2013), 8–19.

Yefu Wang, Kai Ma, and Xiaorui Wang. 2009. Temperature-constrained power control for chip multiprocessors with online model estimation. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09)*. 314–324.

Francesco Zanini, David Atienza, Luca Benini, and Giovanni De Micheli. 2009. Multicore thermal management with model predictive control. In *Proceedings of the 19th European Conference on Circuit Theory and Design*. IEEE Press, 90–95.

Received August 2015; revised February 2016; accepted February 2016