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

HAI WANG, University of Electronic Science and Technology of China
JIAN MA, University of Electronic Science and Technology of China
SHELDON X.-D. TAN, University of California at Riverside
CHI ZHANG, University of Electronic Science and Technology of China
HE TANG, University of Electronic Science and Technology of China

It is challenging to manage the thermal behavior of many-core microprocessors while still keep it running at high-performance since control complexity increases as core number increases. In this paper, 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 DVFS scheme to ensure smooth control behavior and negligible computing performance sacrifice. In order to be scalable to many-core system, 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

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, and He Tang, 2015. Hierarchical Dynamic Thermal Management Method for High-Performance Many-Core Microprocessors. *ACM Trans. Des. Autom. Electron. Syst.* 1, 1, Article 1 (January 2015), 27 pages.
DOI: 0000001.0000001

1. INTRODUCTION

Extremely high operating temperature has 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 cost, and serious reliability issues. Finding economic 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].

This work is supported in part by 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, and in part by an initial startup grant from UESTC.
Author's 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, Chengdu 610054, China. S. X.-D. Tan, Department of Electrical and Computer Engineering, University of California at Riverside, Riverside, California 92521, USA.

DOI: 0000001.0000001

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 to be within the safe range while keeping its computing performance at high level. DTM is usually performed by two categories of methods: task migration and DVFS. 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 multi-core systems [**?**]. Task migration usually leads to high throughput of the system since all cores are running at the maximum speed. But it may suffer from high average temperature problem without any other DTM methods involved.

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 is also used in the dark silicon scenario, which determines the v/f levels for dark silicon chips with temperature as the constraint [**?**; **?**]. DVFS is able to guarantee the thermal safety of the chip, 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 thermal model, thermal sensor information, etc., and outputs the action guidance (for example, how much frequency should be adjust to 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 with multi/many-core thermal systems due to their complex multimodal dynamics [Bartolini et al. 2013]. Recently, model predictive control (MPC) is 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 prediction on the thermal behavior to enhance the control efficiency, it is able to provide more accurate and effective management suggestions. Significant throughput improvement is reported by using MPC comparing with traditional control methods [Bartolini et al. 2013].

Combining MPC with both DVFS and task migration may take the advantages from all the 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 which combines two techniques out of three. Specifically, works which combine task migration and DVFS include the experience based method [Brooks and Martonosi 2001], hybrid method which optimizes performance [Hanumaiah et al. 2011], and hybrid method which optimizes energy consumption [Hanumaiah and Vrudhula 2014; **?**]. Works which contains both MPC and DVFS are presented in [Zanini et al. 2009; Wang et al. 2009; Bartolini et al. 2013]. However, combining MPC with task migration is much harder than combing MPC with DVFS. Recently, a work which combines all three methods is proposed [Ma et al. 2014]. However, this method is only applicable to multi-core microprocessors due to the high overhead introduced in integrating MPC and task migration, especially for many-core systems.

In this paper, 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, and 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 speedup 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 thermal system and electrical circuit system, we can build the thermal model of the microprocesser using thermal equivalant resistors (thermal resistors), thermal equivalant capacitance (thermal capacitors), and thermal equivalant independent current and voltage sources. As a result, similar to the electrical system, the thermal model of a microprocesser with l cores can be expressed as ordinary differential equations like

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

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 also 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 it is the input of the model; $Y(t)$ is the thermal vector with temperature information of the l cores, and it is the output of the model; $L \in \mathbb{R}^{l\times n}$ 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 (1) in continuos time is discretized into the following difference equation by using the 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 (1), A and B_d are formulated using G, C, and B_c, according to the specific numerical integration method used to discretize (1).

For general purpose, thermal model in (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 (2), which is sufficient for

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

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 elimilating the capacitance term from (1), is used to compute the power with the given temperature information, due to its simplicity. However, DTM based on steady-state thermal model will ignore 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 (1) (or the discretized form in (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 current thermal state into consideration and handle transient thermal and power effects, the power it computed still cannot lead to a smooth thermal control. This is mainly because this method lacks of 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 (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 (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 adjacent two steps of (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

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

we can rewrite (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.

[2]The temperature ceiling can be adjust according to real world application, it can be also slightly lower than the highest temperature allowed for absolute safety considerations.

So far, we have obtained the connection between the power input difference and the output core temperatures in (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

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

In this vector, $Y_{max} \in \mathbb{R}^{m \times 1}$ contains the ceiling temperatures of each core. Here we assume 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 (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 consideration, we prefer power distribution not to change drastically. So the extra tuning term $\Delta P_k^T R \Delta P_k$ is added to $(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$, and form

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

as the final function to be minimized over the variable ΔP_k. $R = rI_{N_c \times N_c}$ is tuning matrix with r as the tuning parameter, which determines the weight between the two terms, and can be fine tuned through experiment to adjust practical situations, according to different number of cores, different running applications (different power traces), etc. Also note that Y_k is a function of the unknown variable ΔP_k.

Next, optimization is performed to minimize (8) by taking the first derivative of (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 (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 can be reached without violating the temperature requirements.

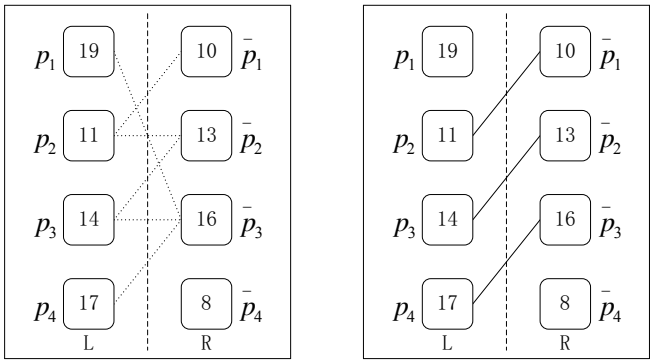
2.3. Task migration and DVFS based on desired power

Next, dynamic thermal management can be performed with the desired power distribution of cores provided by MPC in (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 microprocessor. The basic reason is that DVFS can only lower the power of core but 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 highest voltage level and frequency speed, while MPC suggests this core consume power $\bar{p}_i$ with $\bar{p}_i > p_i$. In this case, nothing can be done by DVFS at the i-th core. But at the same time, there may exist a j-th core consuming $p_j \approx \bar{p}_i$, which must be scaled to a lower power (for example, equal to p_i) by DVFS in order to satisfy the thermal constraint. This leads to performance degradation of the j-th core and will be revealed as lower throughput.

Actually, it is obvious that if we simply swap the loads at the j-th core and the i-th 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 make the migration action according to the matched pairs. This matching process is an assignment problem, and it can be modeled as bipartite graph and solved as a bipartite matching problem. The corresponding bipartite graph can be formulated as $\mathcal{G} = (L, R, E)$, where $L = \{p_1, p_2, \cdots, p_l\}$, $R = \{\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_l\}$, 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 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 load at the i-th core will be moved to the j-th 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 less unmatched powers, less DVFS actions, higher risk and severity of thermal violation, but higher performance of the chip. The proper value of e_{th} should be determined as the one leading to acceptable thermal violation risk and severity. 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 preventing them from connecting to each other, thus avoiding unnecessary task migrations.

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



(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.

3. HIERARCHICAL DYNAMIC THERMAL MANAGEMENT METHOD

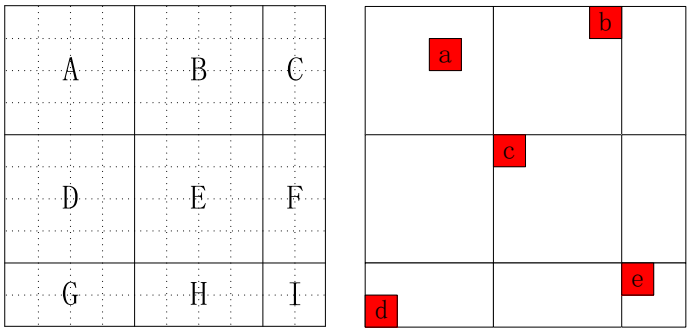
In this section, a new hierarchical DTM method is proposed for high-performance many-core microprocessors. The new technique is based on model predictive control and uses both task migration and DVFS.

It is challenging to perform MPC based task migration on many-core microprocessor, 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 Hungarian algorithm with complexity of $O(n^3)$. In order to handle the many-core system which has a large core number, the new technique clusters the cores into blocks and makes task migration decision in 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 the upper level are processed using 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, i.e., 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



(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.

is the CPU time used to compute the matching in a regular block (note that all edge blocks have 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 less unmatched pairs to be left to the upper level causing less processing time at the upper level.

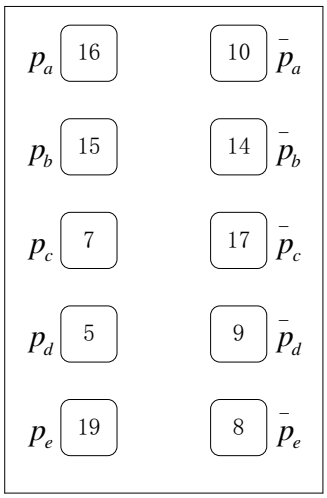
An example of the lower level clustering process is show in Fig. 2 (a). It is a 100-core microprocessor, and the cores are clustered into four 16-core regular blocks (labeled as A, B, D, E). Five edge blocks (labeled as C, F, G, H, I) are also introduced with the number of cores ranging from 4 to 8. Lower level bipartite matching is then performed in each block, with Fig. 1 showing the bipartite matching inside block I of Fig. 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 Fig. 2 (a), the powers p_1 and $\bar{p}_4$ cannot be matched as shown in Fig. 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 Fig. 2 (b), marked in red color.

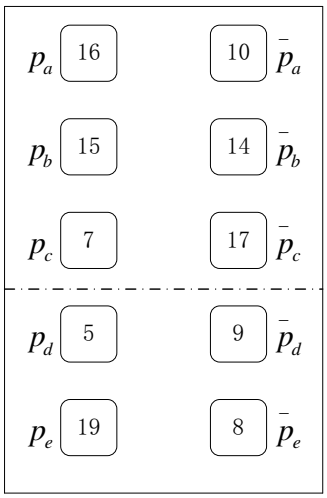
3.2. Higher level task migration among blocks

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.

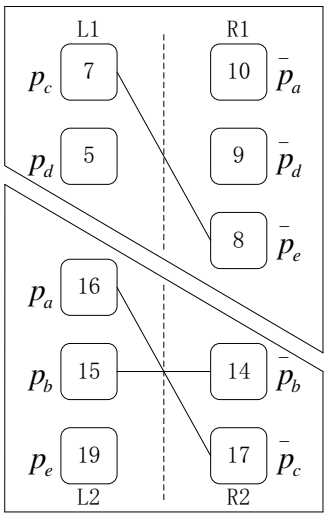
We want to find all the power pairs that can be matched at the upper level, and perform DVFS only on the final unmatched powers. It appears that we can cluster the upper level powers into larger blocks according to their locations similar to what we have done at the lower level. Then we can perform bipartite matching inside the



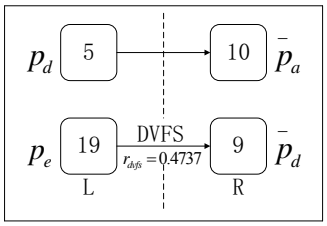
(a) Graph $\mathcal{G}_p$ (edges and weights are not shown for simplicity).



(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.

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 comparing 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 have 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 large computing cost required.

In order to make task migration decision efficiently at higher level, we cluster the upper level powers into blocks in another way, using 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 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 also 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, 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 $V_p = \{p_1, \bar{p}_1, p_2, \bar{p}_2, \cdots, p_m, \bar{p}_m\}$, and E_p contains all connections of vertices in V_p with weights. The weights are defined as $w(p_i, \bar{p}_j) = 1/|p_i - \bar{p}_j|$, $w(p_i, p_j) = 0$, $w(\bar{p}_i, \bar{p}_j) = 0$ for all the i and j. One example of $\mathcal{G}_p$ is shown in Fig. 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}, each with m vertices. The *cost* of the cut is defined as the sum of the weights on the cut set. An example is given in Fig. 3 (b), where the intial cut generates two subsets $V_{p1} = \{P_a, P_b, P_c, \bar{P}_a, \bar{P}_b, \bar{P}_c\}$, and $V_{p2} = \{P_d, P_e, \bar{P}_d, \bar{P}_e\}$.

Then, in order to proceed 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 Fig. 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), etc. 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

[3]Cores in this upper level block are unnecessarily spatially adjacent as the ones in the lower level block.

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 i-th 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 neighbours 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 extremely unbalanced result: one subset contains a lot of vertices (powers), while the other subset contains very little. This will cause problem to 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 $\mathcal{F} = \{f(v_1), f(v_2), \cdots, f(v_{2m})\}$. 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 $v_1, v_2, \ldots, v_{i-1}$ has 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 $\tilde{\mathcal{F}} = \{\tilde{f}(v_1), \tilde{f}(v_2), \cdots, \tilde{f}(v_{2m})\}$, 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 $\{v_1, v_2, \cdots, v_k\}$ to their corresponding other subsets. All procedures above are counted as *one moving action*.

Although the moving action can be performed repeatedly, but previous researches on circuit partitioning show that two to four moving actions are enough to achieve 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 have clustered 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 Fig. 3 (a), (b), and (c). In Fig. 3 (a), all the unmatched powers after the lower level matching (can be

seem in Fig. 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 the Fig. 3 only for simplicity reason. Then, iterative minimum cut algorithm is performed on graph $\mathcal{G}_p$, with the initial cut in dashed line shown in Fig. 3 (b), and the final cut shown in Fig. 3 (c) dividing the powers into two blocks. Next, upper level bipartite matching is performed by formulating new bipartite graph for each block as shown in Fig. 3 (c).

3.3. Final adjustment by DVFS

DVFS method is introduced to do the final adjustment on the unmatched powers from 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 j-th 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 j-th 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 can be achieved by the j-th 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 nearest level which is lower than $\bar{p}_j/p_i$.

The final step performed on the 100 core example is shown in Fig. 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 due to the fact of $p_e > \bar{p}_d$.

It is well known that in 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 which 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 throughput/performance drop. This is a trade-off between DC-DC converter implementation overhead against the chip performance. This trade-off is a general and important problem in many-core architecture, and needs to be further researched in future works.

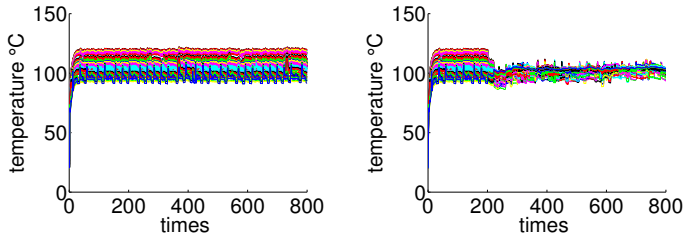
This concludes the new algorithm, and we summarize the whole flow of the new method in Algorithm 1.

4. EXPERIMENTAL RESULTS

The experiments are performed on a Linux server with two 2.90 GHz 8-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 4 different core configurations, from 100 cores (10×10) to 625 cores (25×25). The many-core microprocessors in the experiments are composed of identical Alpha 21264 cores. The dimension of all chips is $10mm \times 10mm \times 0.15mm$. 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

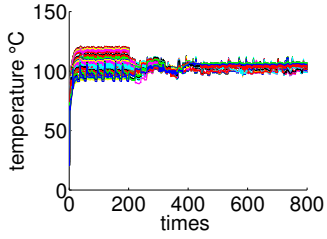
Algorithm 1 Hierarchical dynamic thermal management algorithm

1: Calculate the desired power distribution $\bar{P}(k)$ using MPC with the provided temperature constraint as in (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 cut algorithm.
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.

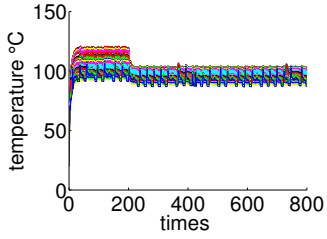


(a) Temperature traces without any DTM method.

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

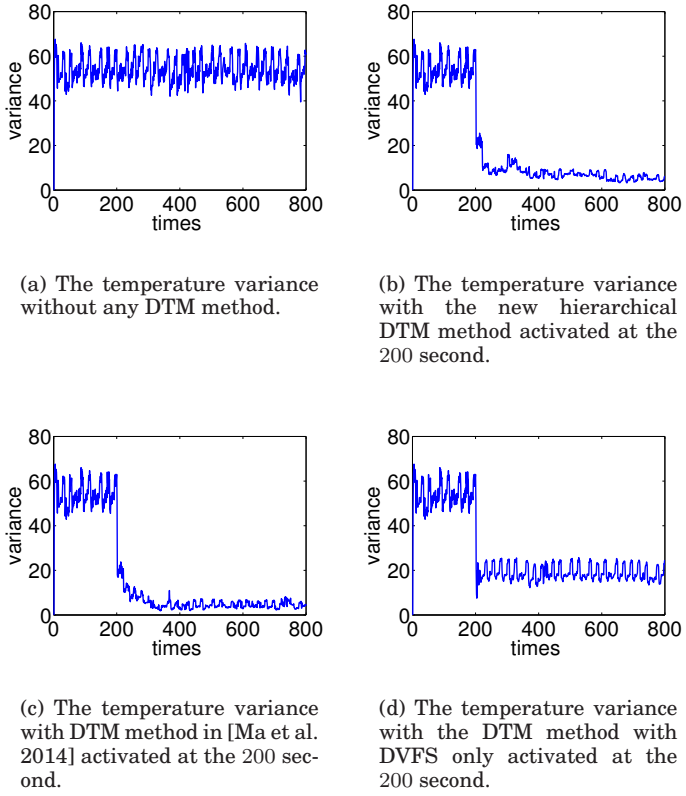


(c) Temperature traces with DTM method in cite [Ma et al. 2014] activated at the 200 second.



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

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



(a) The temperature variance without any DTM method.

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

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

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

DTM method. We use 9 power traces of 9 different SPEC benchmarks. For the power traces of different CPUs, we recycle those 9 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, and this will lead to unrealistic high power density (and extremely high temperature). As a result, we scale the power traces to ensure that all CPUs have similar power density, which leads to a temperature distribution similar to today's multi-core chips. It is achieved by scaling the operating frequency and voltage by the same ratio. For CPUs with different core number, 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 application, 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. The ambient temperature is 20°C. 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.

Table I. Parameters of the new hierarchical method for CPUs with different core configuration.

Configuration	Scale	e_{th}	r
100 cores (10×10)	0.21	0.06	500
256 cores (16×16)	0.08	0.05	2100
400 cores (20×20)	0.052	0.04	3000
625 cores (25×25)	0.033	0.03	3000

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 core number increases. It is usually fine to use much smaller migration period with small number of cores, where computing overhead and migration action overhead (due to core to core communication) are both small. For 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 raises a lot in between the migration intervals and this may cause temperature violation. In such case, we can just perform DVFS inside the migration intervals when needed to enhance the safety.

Overhead of 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 [**?**]. 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 power during 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 multi-core 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 processor. We have used the opensource program, MAGMA V2, provided by the authors of [Hanumaiah et al. 2011] in the experiment. MAGMA can only give 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 Fig. 4 (a). We can see that the temperature of the core ranges from 90°C to 120°C. And the temperature around 120°C will clearly harm the reliability of the chip. The temperature variance among cores is also measured and shown in Fig. 5 (a), it can be seem that the temperature difference among cores is large without any DTM method activated.

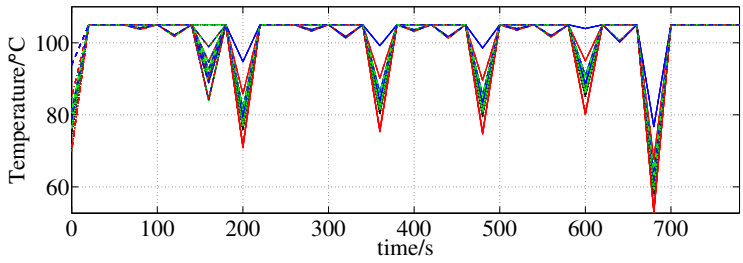
Next, we test the new hierarchical method with the ceiling temperature set as 105°C. The new hierarchical method is activated at the time of 200 second. The corresponding transient temperature traces of the 100-core CPU are shown in Fig. 4 (b), and the corresponding temperature variance is shown in Fig. 5 (b). After the new hierarchical

Table II. The temperature variance among cores in CPUs with different core configuration. 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], and var_n is variance with our new hierarchical method, and var_h is the variance with method in [Hanumaiah et al. 2011].

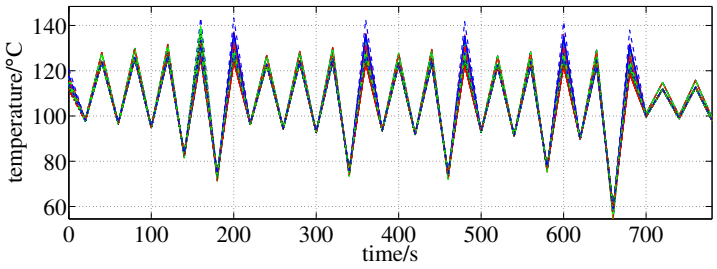
Configuration	var_o	var_d	var_c	var_n	var_h
100 cores (10×10)	54.2	18.8	5.5	7.6	5.9
256 cores (16×16)	50.8	19.3	3.4	6.5	N/A
400 cores (20×20)	52.1	16.7	2.5	4.1	N/A
625 cores (25×25)	50.5	15.9	2.3	4.1	N/A

Table III. The average runtime per second of the DTM methods on CPUs with different core configurations. t_p represents MPC time, t_m means task migration matching time, and t_a denotes for minimum cut partitioning time.

	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



(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 huge impact on temperatures for many-core microprocessors, causing the $105°$C ceiling temperature to be seriously violated. Overshoot problem is also significant comparing to the MPC based techniques.

method being activated, the temperatures of all cores track the given $105°$C ceiling temperature, and the temperature difference is greatly decreased.

For comparison, MPC based method with task migration and DVFS for multi-core 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 second with activating period of 20s. Fig. 4 (c) (d) and Fig. 5 (c) (d) show the corresponding transient temperature traces and temperature variances. The new hierarchical method has similar transient behavior and a slightly higher variance comparing with 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 multi-core chip 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. Fig. 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 zero core to core heat exchange assumption is made. We modified MAGMA program, and plotted the actual temperatures considering heat exchange among cores. The results are shown in Fig. 6 (b). It is clear that the DTM decision made with such assumption is not optimal, and causes significant actual temperature violations. From Fig. 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 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 running at full-speed (if current temperature is lower than the ceiling temperature). All the MPC based algorithms do not have such overshoot problem as shown in Fig. 4, because they are able to predict into the future and make DTM decision which leads to smooth temperature control.

The variance comparison on all CPUs with different number of cores are recorded in Table II, where 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], and var_n is variance with our new hierarchical method, and var_h is the variance with method in [Hanumaiah et al. 2011]. All results for different number of cores are similar to the 100-core example: 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. Method in [Hanumaiah et al. 2011], although can only finish the 100-core case, performs well in variance test, and gives similar results to method in [Ma et al. 2014].

It can be seem from previous comparisons for transient temperature traces and temperature variance, 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 comparing with [Ma et al. 2014] and [Hanumaiah et al. 2011]. Now, we measure the average runtime

Table IV. The average number of instructions per second of one core on CPUs with different configurations. $MIPS_o$ represents the IPS in million (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.

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

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. Runtime of 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, which is dominating the delay. The method in [Ma et al. 2014] shares MPC time t_p and t_m, but do not have partitioning time t_a. The DVFS only method shares only MPC time t_p. Although 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 matrix with correct dimensions filled by random numbers. The computing time of CPU with different number of cores are recorded in Table III. It is obvious that for method in [Ma et al. 2014], the overhead becomes more significant as core number increases. Starting from the 400-core case, for each second of management, this method spends more than one second on computing, which makes it totally inapplicable. For method in [Hanumaiah et al. 2011], the task migration computing time also increases fast 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 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 huge number of cores, the new method is able to make management decision in 4ms for every one second of management. This is only 0.4% of computing time spent on only few number of cores, thus can be considered as neglectable. 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 CPU, 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 million (MIPS) of the core without any DTM method, $MIPS_d$ stands for the average number of IPS in million of the core using DTM with DVFS only, and $MIPS_n$ denotes the average number of IPS in million 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 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, but 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 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 comparing 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 paper, 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 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 safe temperature range, and 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 *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*. 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 *Proc. IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*. 171–182.

David Brooks, Vivek Tiwari, and Margaret Martonosi. 2000. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proc. Int. Symp. on Computer Architecture (ISCA)*. 83–94.

Thidapat Chantem, Sharon Hu, and Robert Dick. 2011. Temperature-Aware Scheduling and Assignment for Hard Real-Time Applications on MPSoCs. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems* 19, 10 (October 2011), 1884–1897.

James Donald and Margaret Martonosi. 2006. Techniques for Multicore Thermal Management: Classification and New Exploration. In *Proc. Int. Symp. on Computer Architecture (ISCA)*. 78–88.

Shantanu Dutt and Wenyong Deng. 1996. A probability-based approach to VLSI circuit partitioning. In *Proc. Design Automation Conf. (DAC)*. 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 *Proc. Int. Conf. on Computer Aided Design (ICCAD)*. 302–309.

C. Fidducia and R. Mattheyses. 1982. A linear-time heuristic for improving network partitions. In *Proc. Design Automation Conf. (DAC)*. 175–181.

Yang Ge, Parth Malani, and Qinru Qiu. 2010. Distributed task migration for thermal management in many-core systems. In *Proc. Design Automation Conf. (DAC)*. 579–584.

Vinay Hanumaiah and Sarma Vrudhula. 2014. Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling. *IEEE Trans. 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 Trans. 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 Trans. on Very Large Scale Integration (VLSI) Systems* 14, 5 (May 2006), 501–513.

Ramkumar Jayaseelan and Tulika Mitra. 2009. A hybrid local-global approach for multi-core thermal management. In *Proc. Int. Conf. on Computer Aided Design (ICCAD)*. 314–320.

M. Kadin, S. Reda, and A. Uht. 2009. Central versus Distributed Dynamic Thermal Management for Multi-Core Processors: Which One Is Better?. In *Proc. IEEE/ACM International Great Lakes Symposium on VLSI (GLSVLSI)*. 137–140.

Guanglei Liu, Ming Fan, and Gang Quan. 2012. Neigbor-Aware Dynamic Thermal Management for Multi-core Platform. In *Proc. European Design and Test Conf. (DATE)*. 187–192.

Jian Ma, Hai Wang, Sheldon Tan, Chi Zhang, and He Tang. 2014. Hybrid Dynamic Thermal Management Method with Model Predictive Control. In *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 Trans. on Circuits and Systems I: Fundamental Theory and Applications* 56, 9 (September 2009), 1994–2007.

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 *Proc. of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 260–270.

Kevin Skadron, Mircea Stan, Wei Huang, Siva Velusamy, Karthik Sankaranarayanan, and David Tarjan. 2003. Temperature-Aware Microarchitecture. In *Proc. Int. Symp. on Computer Architecture (ISCA)*. 2–13.

Yefu Wang, Kai Ma, and Xiaorui Wang. 2009. Temperature-Constrained Power Control for Chip Multiprocessors with Online Model Estimation. In *Proc. Int. Symp. on Computer Architecture (ISCA)*. 314–324.

F. Zanini, D. Atienza, L. Benini, and G. De Micheli. 2009. Multicore Thermal Management with Model Predictive Control. In *Proc. 19th European Conference on Cicuit Theory and Design*. IEEE Press, Piscataway, NJ, USA, 90–95.

Summary of Revision
(paper id: TODAES-2015-P-1153)

6. SUMMARY

We thank Editor in Chief, Associate Editors, and reviewers for your effort of reviewing our manuscript, and providing many insightful suggestions which help us to improve this article. The presentation, derivation, and experiments in this article has been carefully improved to address all questions from the reviewers and editors. Major modifications are highlighted using red fonts in the manuscript.

Before we reply the questions of each reviewer, we would like to highlight the following major modifications:

- In experiment, we have compared our new method with the method in [Hanumaiah et al. 2011], which has similar objective to our method. We have shown and discussed the advantages of our new method comparing with method in [Hanumaiah et al. 2011].
- We have considered the overhead caused by task migration action in the experiment. The power consumption of task migration is also counted. Throughput of the new method is updated accordingly.
- Details of experiment settings are added including how are the SPEC benchmark tasks mapped and scheduled over the cores, and how is the core power scaled when core number increases.
- e_{th} is discussed in details, including how does the value of e_{th} affect the chip performance and reliability. It is also shown how does e_{th} make picking the ceiling temperature as the reference temperature suitable for all occasions by avoiding unnecessary task migration actions.
- All the mentioned presentation issues have been addressed.

Below, we will answer the questions raised by each reviewer. For the completeness, the original comments are *quoted* first, followed by a short explanation of how the comments have been addressed.

6.1. Reply to referee 1

```
This paper proposes a two-level DTM method exploiting task migration based
on MPC decisions (for local management), and DVFS (for global management).
  Although MPC and DVFS are well-known DTM/DPM, this paper defines and
solves a matching problem to use the two methods appropriately.
  Min-cut algorithm and by weighted bypertite matching is well suited for
the presented problem.
  In spite of the interesting approach this paper takes, I would like to
expect the authors to clarify my some questions:
  1) I think power consumption for the task migration should be considered.
I don't know what migration mechanisum the authors targeted, but at least
they should've mentioned it in the paper. If the migration power is not
that small to be ignored, the problem definition should be different.
```

Answer: Thank you for your comment. Power of task migration is mainly caused by the communication between the cores. During such migration time, the corresponding two cores are not processing any tasks, as a result, we assume the power during 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 in this new version of work.

We have made the discussion in the experiment part.

2) The DVFS method in the paper is a per-core DVFS. It is well known that, to perform the per-core DVFS, accompanying DC-DC converter overhead should be taken into consideration. For example, in reality, the cores for the local MPC management in the paper may be powered by only one DC-DC converter.

Answer: Thank you for this valuable suggestion. In the experiments, we use per-core DVFS to demonstrate the performance of the algorithm. However, we add the remark that it is practical to add *DVFS block* which contains a number of spatially adjacent cores which share one DC-DC converter, in order to reduce the implementation overhead of the DC-DC converter with the DVFS efficiency as a trade-off. We think this implementation overhead vs. DVFS efficiency trade-off of DC-DC converter is a general and important problem in many-core architecture, and needs to be researched and solved in future works. We have added discussion in Section 3.3.

3) As the authors uses the task migration scheme, core-consolidation seems to be available too. Sometime it would be better to consolidate some cores and turned off others to save power, which may be more powerful than perform DVFS for all the cores.

Answer: Thank you very much for your comment and suggestion. Core-consolidation technique by turning off unused cores is very effective in saving energy as indicated by the reviewer. Core-consolidation is usually used in works which minimize power consumption according to a throughput (such as Ghasemazar et al. "Minimizing Energy Consumption of a Chip Multiprocessor through Simultaneous Core Consolidation and DVFS"), or deadlines in real-time systems (such as Fu et al. "Utilization-controlled Task Consolidation for Power Optimization in Multi-Core Real-Time Systems"). But the goal in our work is different: we try to maximize throughput (in some senses, maximize power consumption instead of minimize it) with temperature as the constraint. As a result, we think core-consolidation may not be very suitable for our problem scenario.

4) By what does the cost function mean. Please define the problem definition in the more formal way.

Answer: Thank you very much for pointing out this confusion we made. The physical meaning of the cost function $(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$ in (8) is a measurement of the difference between the desired temperature (Y_{ceil}, which is set by user as a target) and the temperature generated by power distribution (unknown, to be determined here). The optimal power distribution is the one leading to $Y_k = Y_{ceil}$, i.e., minimize $(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$. In addition, for practical consideration, we prefer power distribution not to change drastically. So the extra tuning term $\Delta P_k^T R \Delta P_k$ is added to $(Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k)$, and form $F = (Y_{ceil} - Y_k)^T(Y_{ceil} - Y_k) + \Delta P_k^T R \Delta P_k$ as the function to be minimized. We have revised the corresponding part to clarify this confusion.

5) It would be much better to explain clrealy how to determine eth. According to eth, designers can put more weight on performance degradation or thermal runaway avoidance.

Answer: Thank you very much for your suggestion. We have added more discussion on how to determine e_{th} in Section 2.3, including how does the value of eth effect the chip performance and reliability. It is also shown how does eth make picking the ceiling temperature as the reference temperature suitable for all occasions by avoiding unnecessary task migration actions.

6) If the authors can provide details of how to link the setups between the real Linux machine and simulator in experimental work, the experimental work should look more convincible.

Answer: Thank you very much for your advice. At current stage, we have not implemented this work in real Linux machine. We have added more discussions on how can the proposed work be implemented in practical situations. Implementing this new method on real many-core chips is also one of our future research focus.

Plus, in my opinion,if the authors can find more related work w.r.t. temperature/thermal effects and solutions in the chips from the recent conference (i.e., ICCAD, DAC, DATE, ISLPED and ASPDAC), the paper must look better.

Answer: Thank you very much for your suggestion. We have discussed more related works in this revision, such as "Jason Cong and Bo Yuan. ISLPED 2012. Energy-Efficient Scheduling on Heterogeneous Multi-Core Architectures.", "Heba Khdr, Santiago Pagani, Muhammad Shafique, and Jorg Henkel. DAC 2015. Thermal Constrained Resource Management for Mixed ILP-TLP Workloads in Dark Silicon Chips.", "Thannirmalai Somu Muthukaruppan, Mihai Pricopi, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. DAC 2013. Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era.", "Cheng Tan, Thannirmalai Muthukaruppan, Tulika Mitra, and Lei Ju. ASPDAC 2015. Approximation-Aware Scheduling on Heterogeneous Multi-core Architectures.".

P.s., Here comes some minor comments:
i) I don't think using too many "please" in the paper is good.
ii) Please unify the terms "safe temperature ceiling", "temperature ceiling" and "ceiling safe temperature" to one. Or use one symbol.
iii) You don't need to explain Matlab version in the paper, unless R2010a is really special for experimental work.
iv) If caption of Fig. 4 has any hint for reader to understand why the new methods is superior, Fig. 4 should have a reason to be in the paper.

Answer: Thank you. We have fixed them in this revision.

6.2. Reply to referee 2

The journal submission build on the authors' previous conference paper, by adding another hierarchical layer of DTM control, wherein they collect the unmatched tasks (w.r.t. their power to the required power for a core), cluster them using a modified iterative min-cut algorithm, and run the bipartite matching algorithm. The advantage as claimed by the authors is the lower overhead, and the solution is scalable to larger number of cores.
After reviewing the authors I've some major concerns regarding the objective of the proposed DTM and the results. They are listed below:
1) In the MPC optimization for DVFS, why the reference is set to be the max temperatures of cores? I understand that the authors suggest that the max temperatures could be set according to the desired objective, either

performance/energy saving, but in reality it is very hard to do. I would like the authors to show examples of how different objectives could be mapped to maximum temperature. Otherwise, the goal of matching the maximum temperatures will not be a practical solution.

Answer: Thank you very much for the comment and suggestions.

We set the reference as the maximum temperature because the goal of this work is to maximize performance (throughput) with temperature as the constraint in high-performance processor. We are actually trying to maximize power (which is closely connect to throughput), and do not consider energy saving. This setting is similar to the paper Hanumaiah et al. "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors", IEEE TCAD, 2011 listed by the reviewer in the next question. It shares similar objective as our work, and also set the reference as the maximum temperature allowed.

In reality, there are times that the loads of most cores are not heavy enough to result the maximum temperature. In such case, we also do not need to change the reference temperature (just keep the reference temperature as the maximum one allowed). Because the current power of the i-th core p_i will be lower than the desired power $\bar{p}_i$ from MPC, by setting the proper e_{th}, it will naturally lead to zero DVFS and task migration actions without any performance degradation.

In summary, we think setting the reference temperature to be the maximum temperature is general enough to achieve the objective of our work: maximize throughput (power) with thermal constraints. We have also added this discussion in Section 2.3 as a function of e_{th}.

2) If we look at the results, they are only compared with either their own previous implementation, or with DVFS only result. The hybrid DTM is not new. Several authors have explored the approach, e.g.

(i) Khdr et al. "Thermal Constrained Resource Management for Mixed ILP-TLP Workloads in Dark Silicon Chips", DAC 2015.

(ii) Hanumaiah et al. "Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors", IEEE TCAD, 2011.

It is hard to judge an approach without comparing the results to state-of-the-art solutions.

Answer: Thank you very much for your suggestion and providing the references. Because our work is not targeting dark silicon chips, and reference (ii) shares similar goal of our work, i.e., maximize throughput with thermal constraint, we compared the new method with reference (ii). From the experiments, it can be seem that our work has advantages over reference (ii) in three aspects. First, our algorithm contains a hierarchical structure to reduce the assignment problem solving time in task migration. But reference (ii) does not have such a structure, and according to their paper, suffers the $O(nq)^3$ complexity (which is similar 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 (ii). When the core/task number increases, such $O(n^6)$ complexity will make the task migration too time consuming to be used. Second, our algorithm employs MPC technology to prevent the overshoot problem and achieved smooth temperature control. In (ii), the zero-slack policy is used for temperature/power control: when current temperature is not exactly the same as the target/ceiling temperature, the core is either turned off (if current temperature is higher than the ceiling temperature) or running at full-speed (if current temperature is lower than the ceiling temperature). Such control scheme may lead to serious overshoot problems as shown in the experiment. Third,

we considered the heat exchange among cores which makes the DVFS and task migration decision accurate. In (ii), each core is assumed to be thermally isolated with all other cores, and it can only exchange heat with TIM. Such assumption may hold for 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 as shown in the experiment.

We have added the experiment results and discussions in the experiment part.

3) The authors haven't considered the migration overhead, which is much more than DVFS. The reason DVFS is preferred is for it's very low over head. Also, why do other use 20 s migration period, while the current migration interval in Linux OS is around 200 ms? It's useful to mention the overhead faced with actual migration, and how authors are planning to adopt the proposed scheme in a real OS.

Answer: Thank you very much for pointing out the overhead problem. We have considered the migration overhead in this revision. The normal task migration action overhead is around 10^6 cycles, which counts for around 1ms for a 1GHz processor [**?**]. 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 counted this in the throughput measurement. We also considered this overhead in MPC, and set the power consumption during the task migration time to be the previous task average power processed on the core, which is pessimistic in order to guarantee the safety of the chip.

We used 20s as the migration period just to minimize the overhead effect from task migration (overhead from both computing and migration action) when core number increases. It is usually fine to use much smaller migration period with small number of cores, where computing overhead and migration action overhead (due to core to core communication) are both small. For 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 raises a lot in between the migration intervals and this may cause temperature violation. In such case, we can just perform DVFS when needed inside the migration intervals to enhance the safety.

We have revised the experiment section and added more discussions.

Minor comments:

1) The authors need to recheck the notations in some equations, e.g. 'L' is suddenly missing from (2) to (4).

2) In (5), y(k) should be Y(k).

3) Subscripts in Table 2, 3, and 4 are not intuitive. It's either better to put a legend right near the table, or find subscripts that are intuitive.

4) Is the runtime computed by adding computation of all matching operations in serial or in parallel?

Answer:

1) Fixed, thank you.

2) Fixed, thank you.

3) We have added the explanation in the captions, thank you.

4) Runtime of matching operation 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, which is dominating the delay. We have added this in the experiment section.

6.3. Reply to referee 3

This paper presents a hierarchical dynamic thermal management method with task migration and DVFS based on a model predictive control (MPC). The authors claim that the proposed hierarchical approach enhances the scalability of the thermal management for the many-core systems. Well-defined problem statement and solution algorithm are well appreciated.

First, I have a concern about the experimental setup. The authors have used SPEC 2000 benchmarks for the multicore system. To my understanding, SPEC 2000 benchmark was not developed for the multicore system. The geometrical distribution of the temperature on the multicore systems is strongly dependent on the task assignment and scheduling result over the cores. It is needed to explain how the benchmark tasks are distributed and scheduled over the cores where it is not designed for the multicore systems.

Answer: Thank you very much for your comment. In this work, the many-core system is composed of identical Alpha 21264 cores. We use Wattch simulator to get the power traces of different applications (tasks) of SPEC 2000 benchmarks. Then, we randomly assign one task to one core as an initial task assignment. Next, task assignment and scheduling actions are determined by our newly proposed DTM method. We have added these in the experiment section.

I also have a question about the following statement:

Because the size of the core scales as the core number increases, and this will lead to unrealistic high power density (and extremely high temperature). As a result, we scale the power traces to ensure that all CPUs have similar power density.

Then, how about the operating frequency and performance of the unit cores? What is the exact meaning of the power scaling with similar power density? It is unclear that the authors adopt finer manufacturing process to put more number of cores into the same silicon area. If so, the authors should use different power and thermal model parameters. We probably dont need serious thermal management method if we can maintain power density of the systems with the more number of cores without performance degradation.

Answer: Thank you very much for your question. Because it is hard to accurately predict the impact of the future manufacture process on the power/thermal performance of the many-core systems, we make the prediction (assumption) by assuming the average temperature (power density) does not change much from today's technology. It is achieved by assuming operating frequency and voltage scales at the same ratio. We have revised experiment section to include such information.

The authors also should clarify the generality of the manually tuned parameters. It is better to present the parameter explicitly.

Answer: Thank you very much for your suggestion. There are two parameters in our algorithm which need to be tuned manually: e_{th} for the task migration matching process, and r in (8) for MPC. Detailed discussion of e_{th} is added in Section 2.3 in this revision. Explanation of r is also revised before (8). The optimal values of the two parameters 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 application, 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. We have added this discussion in experiment section.

Next, more importantly, the experimental result is failing to show the advantage of the proposed method. The authors claim as follows:
What really shines for the new hierarchical method is the scalability and small overhead for many-core microprocessors comparing with [Ma et al. 2014].
The authors use two baselines for the comparison in the experiment - the MPC based method with DVFS and task migration [Ma et al. 2014] for multi-core microprocessor and the MPC based method with DVFS only. In the computational overhead analysis result in Table 3, the proposed method shows 3.88 ms overall computation time for the 625 core case and the method from [Ma et al. 2014] shows 8.63 s. The authors claim on the scalability of the proposed method is based on this result. However, the other competitor - DVFS only - only requires 0.887 ms computation time where it loses only 3.2% of performance compared to the proposed method in Table 4. In short, the proposed method enhance around 3% of performance with about 4 times longer computation overhead. Im not sure this result really supports the authors claim on the scalability of the proposed method.

Answer: Thank you very much for your comment. The throughput enhancement of the new method compared to DVFS only method is highly dependent on the running applications. If there exists low temperature cores (or even idle cores), then the throughput improvement of the new method will be much more significant comparing to the DVFS only method, because the low temperature cores can be fully used in the migration process to reduce the DVFS actions of our new method. The throughput of the new method is actually quite close to $MIPS_o$, which is considered as the ideal throughput since it is obtained without any DTM performed by ignoring the temperature violation. For the computing overhead side, although the overhead of the new method is 4 times longer than the DVFS only method, both of them can be neglected because both of them are too small to have any impact on the throughput. We have added this remark at the end of the experiment section.