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

***Abstract*—It is challenge to manage the thermal behavior of many-core microprocessors while still keep it running at high-performance as 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 cores are spatially clustered into blocks, and the hierarchical control scheme is designed with two levels. At the lower level, local task migration is used to match current load distribution with the optimal distribution calculated by MPC. At the upper level, global task migration is used with the unmatched loads from the lower level. An iterative minimum cut algorithm can be used to lower the overhead if the load number is large at the upper level. Finally, DVFS is applied to regulate the power of the unmatched loads. Experiments show that the new method is highly scalable to many-core microprocessors with little computing performance compromises comparing with traditional methods.**

I. 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 [1].

Dynamic thermal management (DTM) method is one effective technique to improve the thermally related performance of microprocessors [2]. It controls the running behavior of the microprocessor to make sure its temperature is within the safe range while keeping its computing performance. DTM methods are usually executed by two means of methods: task migration method and DVFS method. Task migration method [3], [4], [5], [6], [7], [8] switches tasks among different cores to lower the peak temperature of the chip. It leads to high throughput of the system since all cores are running at the maximum speed. But it may suffer from average heating problem without any other DTM method involved. DVFS method [9], [10], [11] controls voltage and operating frequency speed to adjust the heat dissipation of the chip. It is able to guarantee the thermal safty of the chip, but performance is sacrificed due to the frequency scaling.

In order to guide the DVFS and/or task migration, a control scheme is usually employed except for experience based or heuristic based methods. The controler takes the temperature information from physical thermal sensor and/or software thermal sensor, and output the action guidence for DTM techniques. Many DTM methods are based on traditional control methods [12]. But these methods do not suit very well with multi/many-core thermal systems due to [13]. Recently, model predictive control (MPC) is introduced in DTM [14], [15], [13]. It uses the thermal model of the microprocessor and makes prediction on the thermal behavior to enhance the control efficiency. Significant throughput improvement is reported by using MPC comparing with traditional control methods.

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, and gauranteed thermal safty. There are a number of works which combines two techniques out of three. Specially, works which combine task migration and DVFS includs the experience based method [16], hybrid method which optimizes performance [17], and hybrid method which optimizes energy consumption [18]. Works which combine MPC and DVFS are presented in [14], [15], [13]. Combining MPC with task migration is much harder than combing MPC with DVFS. Recently, a work which combines all three methods is proposed [19]. However, this method is only applicable to multi-core microprocessors due to the high overhead introduced in MPC and task migration integration, 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 using 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 mantain the high performance of the chip without violating the thermal constraint.

II. Model Predictive Control Based Dynamic Thermal Management

In this section, we will introduce model predictive control based dynamic thermal management method.

Thermal model of a microprocesser with l cores is expressed as

$$\begin{aligned} T(k+1) &= AT(k) + BP(k), \\ Y(k) &= CT(k), \end{aligned} \tag{1}$$

where $T(k) \in \mathbb{R}^n$ is the thermal vector representing temperatures of n blocks of the processor, which includes l cores (with $l < n$) and also boundary condition nodes; $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times l}$ contains thermal dynamic information determined by the thermal conductance, thermal capacitance, and the topology of the processor; $P(k) \in \mathbb{R}^l$ is the power vector of l cores at time k, and is the input of the model; $Y(k)$ is the thermal vector of l cores, and is the output of the model; $C \in \mathbb{R}^{l\times n}$ is the output selection matrix, which selects the l core temperatures from $T(k)$.

Model predictive control is recently used to guide the dynamic thermal managment process, because it is able to calculate the input adjustment needed in order to track a user defined output for a system in the form of (1).

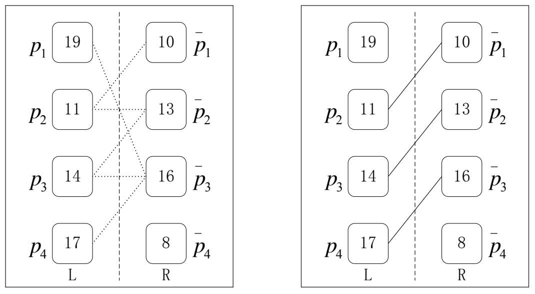
In order to maximize the performance of the processer under a thermal constraint, the highest temperature allowed (called safe temperature ceiling) for each core T_{max} is usually provided as the user defined output to be tracked.[1] Then, MPC is able to calculate the power adjustment $\Delta P(k) = [\Delta p_1(k), \Delta p_2(k), \cdots, \Delta p_l(k)]$, where $\Delta p_i(k)$ is the power adjustment for the i-th core at the k-th step, for all l cores which leads to the desired temperature ceiling. Denote the orginal power as $P(k) = [p_1(k), p_2(k), \cdots, p_l(k)]$, where $p_i(k)$ is the power of the i-th core at the k-th step, then the desired power $\bar{P}(k) = [\bar{p}_1(k), \bar{p}_2(k), \cdots, \bar{p}_l(k)]$ at the k-th step which leads to temperature ceiling can be easily calculated as

$$\bar{P}(k) = P(k) + \Delta P(k). \tag{2}$$

Please note that from now on, we may use P for $P(k)$, p_i for $p_i(k)$, and $\bar{p}_i$ for $\bar{p}_i(k)$, etc., for simplicity in expression without introducing confusions. Superior to traditional controlers, MPC is able to predict into the future and as a result, the power adjustment ΔP it offered leads to smoother temperature control than traditional methods and the resulted throughput improvement has also been verified [20]. Comprehensive MPC introdution and discussion are provided in [21].

Next, dynamic thermal management methods can be executed with the desired power distribution of cores provided by MPC in (2). DVFS can be integrated with MPC easily by lowering the frequency and voltage level of cores to adjust theirs powers to match the desired power distribution calculated 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, but MPC suggests this core consume power $\bar{p}_i$ with $\bar{p}_i > p_i$. In this case, there is nothing can be done by DVFS at the i-th core, but at the same time, there may exist a j-th core which consumes $p_j = \bar{p}_i$, but must be scaled to a lower power level $\bar{p}_j = p_i$ 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 throughtput.

[1]Please note that the safe temperature ceiling can be adjust according to real world application, it can be also slightly lower than the highest temperature allowed for absolute safty considerations.



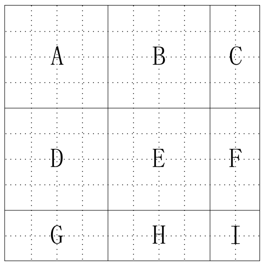
(a) The weighted bipartite graph with the threshold of 3.

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

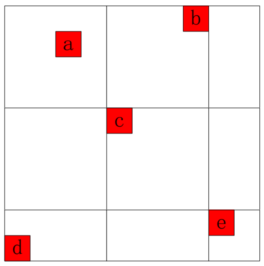
Fig. 1. An example showing the weighted bipartite matching.

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 (except for the newly introduced swapping overhead). As a result, task migration can be performed first by matching current powers P and desired powers $\bar{P}$ into pairs. This 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 $L = \{p_1, p_2, \cdots, p_l\}$, $R = \{\bar{p}_1, \bar{p}_2, \cdots, \bar{p}_m\}$, and E contains *partial* edges between verteces 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 best pairs, then load at the i-th core will be moved to the j-th core). There may exist some unmatched pairs left after the bipartite match, which will be processed by DVFS as introduced later.

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



(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.

III. 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, a modified iterative minimum cut algorithm [22] 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 gaurantee all powers are matched. 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.

A. Lower level task migratioin within blocks

First, we cluster the cores into blocks. As the first step, we can simply cluster the cores according to their location, i.e., we simply group the spatially clustered cores into a block. This introduces no overhead at all and forms the lower level of the whole algorithm. We usually form blocks in square shapes called *regular blocks*, but rectangular or smaller square shaped blocks may appear at the edges, and they are called *edge blocks*.

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

The number of cores inside each block can be tuned to achieve a smaller overhead 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, resulting in less unmatched pairs to be left to the upper level causing less processing time at the upper level.

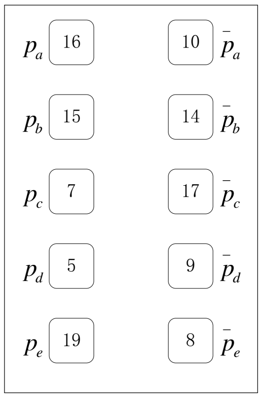
An example of the cluster is show in Fig. 2 (a). It is a 100-core microprocessor, and the cores are clustered into four 16-core regular blocks (labeled as A, B, D, E). Five edge blocks (labeled as C, F, G, H, I) are also introduced with the number of cores ranging from 4 to 8. Fig. 1 also demonstrates the lower level bipartite matching process, specially, it is the bipartite matching inside block I of Fig. 2 (a).

Obviously, it is not adequent 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 stage, because the unmatched powers in one block may find a good match in another un-matched power from other block. This will avoid a lot of DVFS actions and reduce 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.

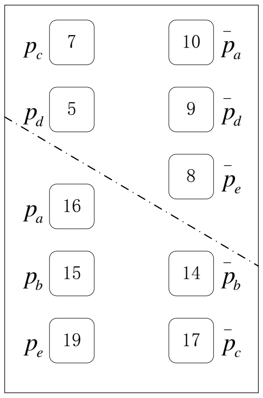
B. Higher level task migratioin among blocks

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.

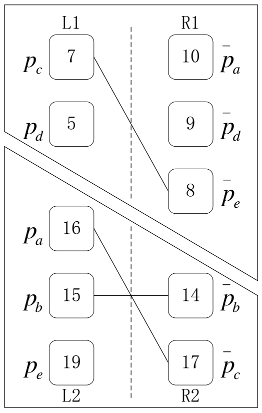
We want to find all the power pairs that can be matched at the upper level, and perform DVFS only on the final unmatched powers. It appears that we can cluster the upper level powers into larger blocks according to their locations similar to what we have done at the lower level, then we can perform bipartite matching inside the new blocks and form even larger blocks using previously unmatched ones until the full chip finally becomes one block. However, experiments show that a much smaller ratio (below 25%) of powers at the higher level can be finally matched than that of the lower level (above 60%). So doing such clustering leads to an even lower match ratio inside each new block, causing the block size to increase very slowly. It is even possible that the block will never grow to the size of the full chip because there may have a lot of powers that cannot be eventually matched, so



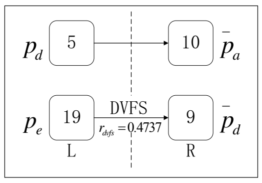
(a) Graph $\mathcal{G}_p$(weight is not shown).



(b) The partitioning result.



(c) Mathcing after partitioning.



(d) DVFS or task migration directly after last matching.

Fig. 3. partition.

before the block becomes to the chip size, biparted matching cannot be performed inside this block due to extremely large overhead.

In order to make matching 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) similar to the biparted graph using all the upper level powers. Then, we use minimum cut algorithm to divide the graph vertically into two groups, and each group is a new block. 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, and if there are unmatched powers left after the bipartite matching inside a block, these unmatched powers also cannot be matched with powers from other blocks.

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 similar to the one used in network partition [22]. First, we built a new graph $\mathcal{G}_p = (V_p, E_p)$ using all the upper level powers, Where vertex set $V_p = \{p_1, \bar{p}_1, p_2, \bar{p}_2, \cdots, p_m, \bar{p}_m\}$, and edge set E_p contains all connections of verteces 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. What we need to solve here is a minimum cut problem on the graph $\mathcal{G}_p$.

As an iterative algorithm, we make the initial cut first by partition V_p into two subsets V_{p1} and V_{p2}, each with m verteces. The *cost* of the cut defined as the sum of the weights on the cut set. The iteration will be proceeded by moving the correct verteces from one subset to the other one. And in order to determine which verteces to be moved, we need to measure the cut cost changes of a moving action. In order to do that, 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. And the following *gain* function $f(v)$

$$f(v) = \sum_{n_i \in E(v)} w(n_i) - \sum_{n_j \in I(v)} w(n_j). \tag{3}$$

measures the immediate decrease in cut cost if v is moved to the other subset. For the i-th iteration, the best vertex is chosen to be moved to the other subset, and this vertex is denoted as v_i which will be locked in that subset, and the gain of all its neighbers will be updated. In our case, using the best vertex in every iteration may result in extremely unbalanced result: one subset contains a lot of verteces (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 overhead. As a result, we introduce a *balance threshold* to the iterative minimum cut algorithm: if moving the best vertex from subset A to B will violate 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 $F = \{f(v_1), f(v_2), \cdots, f(v_{2m})\}$. Now, we can just take the prefix sum of F, and form the prefix sum sequence $\tilde{F} = \{\tilde{f}(v_1), \tilde{f}(v_2), \cdots, \tilde{f}(v_{2m})\}$. Assume $\tilde{f}(v_k)$ is the largest element in $\tilde{F}$, we can perform the actual action by moving $\{v_1, v_2, \cdots, v_k\}$ to their corresponding other subsets. The above is counted as one moving action. Although the moving action can be performed repeatedly, but previous researches show that two to four moving actions are enough to achieve local minimum [22], [23]. 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 in red) are collected to form the graph $\mathcal{G}_p$. Please note that *all* verteces 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 $\mathcal{G}_p$, and the cut is performed using the dot dashed line shown in Fig. 3 (b), clustering 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).

C. 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 an 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 will be lower than the required temperature ceiling. 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 processor without violating the temperature constraint. It is also noted that DVFS makes discrete voltage/frequecy 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 since $p_e > \bar{p}_d$.

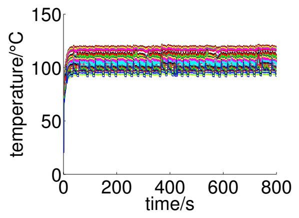
This concludes the new algorithm, and we summarize the whole flow of the new method in Algorithm 1.

IV. Experimental Results

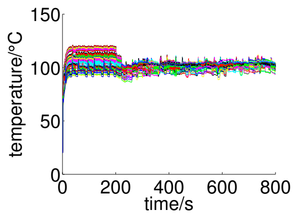
The experiments are performed on a Linux server with two 2.90 GHz 8-core 16-thread CPUs and 64GB memory. The new hierarchical method is implemented using MATLAB R2010a, and HotSpot [24] is used to build the thermal model based on the many-core microprocessors with 4 different core configurations, from 100 cores (10×10) to 625 cores (25×25).

Algorithm 1 Hierarchical dynamic thermal management algorithm

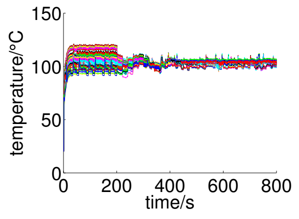
```
1: Calculate the desired power distribution $\bar{P}(k)$ using MPC
   with the provided temperature constraint as in (2).
2: Cluster the adjcent cores into blocks.
3: For each block, build its own bipartite graph $\mathcal{G} = (L, R, E)$
   using the corresponding part of $P(k)$, $\bar{P}(k)$, and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



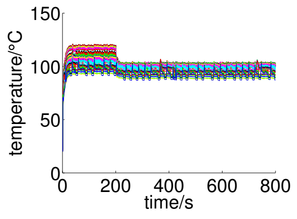
(a) Temperature traces without any DTM method.



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

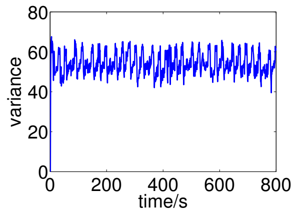


(c) Temperature traces with DTM method in [19] activated at the 200 second.

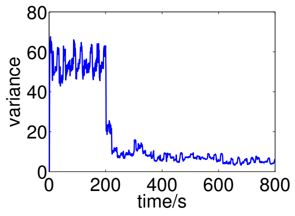


(d) Temperature traces with the DTM method in DVFS only activated at the 200 second.

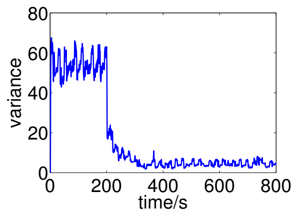
Fig. 4. Transient temperature traces of the 10×10 CPU with different DTM methods. Lines with different colors represents temperature traces of different cores.



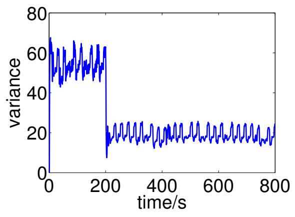
(a) The temperature variance without any DTM method.



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



(c) The temperature variance with DTM method in [19] activated at the 200 second.



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

Fig. 5. Transient variances among 100 cores in the 10×10 CPU with different DTM methods.

The dimension of all chips is $10mm \times 10mm \times 0.15mm$. Wattch [25] is used to generate the power by running SPEC benchmarks [26]. We use 9 power traces on 9 different SPEC benchmarks. For the power traces of those cases, 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 will lead to unrealistic high power density (and extremely high temperature), we scale the power traces to ensure that all CPUs have similar power density. And for CPUs with different core number, the threshold e_{th} in the task migration process and the tuning parameter r (please see [21] for details) in MPC are manually tuned for each CPU to ensure the temperature tracking quality. All these parameters are shown in Table I. The DVFS and task migration activating period is set to be every $20s$, and 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 iterative minimum cut algorithm if the number of vertices in graph $\mathcal{G}_p$ is more than 240. For comparison, we have implemented two other MPC based method, the MPC based method with DVFS and task migration [19] for multi-core microprocessor and the MPC based method with DVFS only for many-core microprocessor.

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

TABLE I
PARAMETERS OF THE NEW DISTRIBUTED METHOD FOR CPUS WITH DIFFERENT CORE CONFIGURATION.

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

TABLE II
THE TEMPERATURE VARIANCE AMONG CORES IN CPUS WITH DIFFERENT CORE CONFIGURATION.

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

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 safe temperature ceiling set as $105°C$. The new hierarchical method is activated at the time of 200 second. The corresponding transient temperature traces of the 100 CPU are shown in Fig. 4 (b), and the corresponding temperature variance is shown in Fig. 5 (b). After the new distributed method being activated, the temperatures of all cores track the given $105°C$ temperature ceiling, and the temperature difference is greatly decreased.

For comparison, MPC based method with task migration and DVFS for multi-core microprocessors in [19] and the MPC based method with DVFS only are tested. The temperature ceilings 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 [19] (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 [19] in terms of transient temperature control and variance among cores.

The variance comparison on all CPUs with different number of cores are recorded in Table II, where var_o is the variance of transient temperature traces without any DTM method, var_d is variance of transient temperature traces with DTM using DVFS only, var_c is variance of transient temperature traces with certralized DTM in [19], and var_n is the variance of transient temperature traces with our new distributed method. All results for different number of cores are similar to the 100-core example: method in [19] slightly outperforms the new hierarchical method, while the DVFS only MPC based method has significantly larger variance among cores.

It can be seem from previous comparisons for transient

TABLE III
THE AVERAGE RUNTIME PER SECOND OF THE DTM METHOD ON CPUS WITH DIFFERENT CORE CONFIGURATIONS.

	new hierarchical method				DTM method in [19]			DTM method with DVFS only	
Configuration	$t_p(10^{-4}s)$	$t_m(10^{-4}s)$	$t_a(10^{-4}s)$	$t_{all}(10^{-4}s)$	$t_p(10^{-4}s)$	$t_m(s)$	$t_{all}(s)$	$t_p(10^{-4}s)$	$t_{all}(10^{-4}s)$
100cores(10×10)	1.58	1.63	0	3.21	1.49	0.01	0.01	1.60	1.60
256cores(16×16)	2.85	19.26	1.58	23.7	2.93	0.45	0.45	2.80	2.80
400cores(20×20)	4.88	7.77	3.27	15.9	5.38	1.90	1.90	5.29	5.29
625cores(25×25)	9.09	12.27	17.49	38.8	8.27	8.63	8.63	8.87	8.87

TABLE IV
THE AVERAGE NUMBER OF INSTRUCTIONS PER SECOND OF ONE CORE ON CPUS WITH DIFFERENT DISTRIBUTION.

Configuration	$MIPS_o$	$MIPS_d$	$MIPS_n$
100cores(10×10)	290.8	279.6	283.4
256cores(16×16)	210.2	202.9	208.2
400cores(20×20)	182.1	174.7	179.9
625cores(25×25)	156.5	150.2	155.2

temperature traces and temperature variance, the difference between our new distributed method and the method in [19] is small. What really shines for the new hierarchical method is the scalability and small overhead for many-core microprocessors comparing with [19]. Now, we measure the average runtime per second excution 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 method in [19] 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. The computing time of CPU with different number of cores are recorded in Table V. It is obvious that for method in [19], the overhead becomes more significant as core number increases and starting from 400-core case, for each second of management, this method spends more than one second on computing, which makes it totally unapplicable. 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 management. This is only 0.4% of computing time spent for 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 will be shown later.

Finally, we measure the performance of different many-core CPUs using different DTM methods. Instructions per second (ISP) is used to estimate CPU performance. Since the method in [19] show significant overhead and cannot complete the management decision in time for many-core CPU with core number large than 400, it is 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 introduced even for a huge number of cores. The new method also outperforms the DVFS only method in terms of CPU performance. Although it is slightly larger in overhead, but the time spent in task migration decision making reduces the DVFS activation times, and brings great overall benifits.

V. 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 method 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 in 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 both hybrid MPC based DTM method and MPC based DTM method with only DVFS with higher computing performance of many-core microprocessors.

REFERENCES

[1] D. Brooks, R. Dick, R. Joseph, and L. Shang, “Power, thermal, and reliability modeling in nanometer-scale microprocessors,” *IEEE Micro*, vol. 27, no. 3, pp. 49–62, May-June 2007.

[2] J. Donald and M. Martonosi, “Techniques for multicore thermal management: Classification and new exploration,” in *Proc. Int. Symp. on Computer Architecture (ISCA)*, June 2006, pp. 78–88.

[3] M. Powell, M. Gomaa, and T. Vijaykumar, “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)*, October 2004, pp. 260–270.

[4] Y. Ge, P. Malani, and Q. Qiu, “Distributed task migration for thermal management in many-core systems,” in *Proc. Design Automation Conf. (DAC)*, June 2010, pp. 579–584.

[5] T. Chantem, S. Hu, and R. Dick, “Temperature-aware scheduling and assignment for hard real-time applications on mpsocs,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 10, pp. 1884–1897, October 2011.

[6] G. Liu, M. Fan, and G. Quan, “Neigbor-Aware Dynamic Thermal Management for Multi-core Platform,” in *Proc. European Design and Test Conf. (DATE)*, March 2012, pp. 187–192.

TABLE V

THE AVERAGE RUNTIME PER SECOND OF THE DTM METHOD ON CPUS WITH DIFFERENT CORE CONFIGURATIONS.

Configuration	$t_n(10^{-4}s)$	$t_c(s)$	$t_d(10^{-4}s)$
100cores(10×10)	3.21	0.01	1.60
256cores(16×16)	23.7	0.45	2.80
400cores(20×20)	15.9	1.90	5.29
625cores(25×25)	38.8	8.63	8.87

[7] R. Ayoub and T. Rosing, “Predict and act: Dynamic thermal management for multi-core processor,” in *Proc. Int. Symp. on Low Power Electronics and Design (ISLPED)*, August 2009, pp. 99–104.

[8] T. Ebi, M. Al Faruque, and J. Henkel, “TAPE: thermal-aware agent-based power economy for multi/many-core architectures,” in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, 2009, pp. 302–309.

[9] K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, “Temperature-aware microarchitecture,” in *Proc. Int. Symp. on Computer Architecture (ISCA)*, 2003, pp. 2–13.

[10] R. Jayaseelan and T. Mitra, “A hybrid local-global approach for multi-core thermal management,” in *Proc. Int. Conf. on Computer Aided Design (ICCAD)*, 2009, pp. 314–320.

[11] A. Mutapcic, S. Boyd, S. Murali, D. Atienza, G. De Micheli, and R. Gupta, “Processor speed control with thermal constraints,” *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, vol. 56, no. 9, pp. 1994–2007, September 2009.

[12] M. Kadin, S. Reda, and A. Uht, “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)*, 2009, pp. 137–140.

[13] A. Bartolini, M. Cacciari, A. Tilli, and L. Benini, “Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-predictive controller,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 1, pp. 170–183, January 2013.

[14] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, “Multicore thermal management with model predictive control,” in *Proc. 19th European Conference on Cicuit Theory and Design*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 90–95.

[15] Y. Wang, K. Ma, and X. Wang, “Temperature-constrained power control for chip multiprocessors with online model estimation,” in *Proc. Int. Symp. on Computer Architecture (ISCA)*, 2009, pp. 314–324.

[16] D. Brooks and M. Martonosi, “Dynamic thermal management for high-performance microprocessors,” in *Proc. IEEE Int. Symp. on High-Performance Computer Architecture (HPCA)*, Jan. 2001, pp. 171–182.

[17] V. Hanumaiah, S. Vrudhula, and K. Chatha, “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*, vol. 30, no. 11, pp. 1677–1690, November 2011.

[18] V. Hanumaiah and S. Vrudhula, “Energy-efficient operation of multicore processors by DVFS, task migration, and active cooling,” *IEEE Trans. on Computers*, vol. 63, no. 2, pp. 349–360, February 2014.

[19] J. Ma, H. Wang, S. Tan, C. Zhang, and H. Tang, “Hybrid dynamic thermal management method with model predictive control,” in *IEEE Asia Pacific Conference on Circuits and Systems*, November 2014.

[20] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, “Thermal-aware system-level modeling and management for multi-processor systems-on-chip,” in *Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS)*, 2011, pp. 2481–2484.

[21] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB*. Springer, 2009.

[22] C. Fidducia and R. Mattheyses, “A linear-time heuristic for improving network partitions,” in *Proc. Design Automation Conf. (DAC)*, 1982, pp. 175–181.

[23] S. Dutt and W. Deng, “A probability-based approach to vlsi circuit partitioning,” in *Proc. Design Automation Conf. (DAC)*. ACM, 1996, pp. 100–105.

[24] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, “HotSpot: A compact thermal modeling methodology for early-stage VLSI design,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 5, pp. 501–513, May 2006.

[25] D. Brooks, V. Tiwari, and M. Martonosi, “Wattch: A framework for architectural-level power analysis and optimizations,” in *Proc. Int. Symp. on Computer Architecture (ISCA)*, 2000, pp. 83–94.

[26] J. L. Henning, “SPEC CPU 2000: Measuring CPU performance in the new millennium,” *IEEE computer*, vol. 1, no. 7, pp. 28–35, July 2000.