Journal of
Low Power Electronics
Vol. 10, 1–16, 2014

Distributed Task Migration in a Homogeneous Many-Core System for Leakage and Fan Power Reduction

Yang Ge[1,*], Yukan Zhang[2], and Qinru Qiu[2]

[1] *Broadcom Corporation, 5300 California Avenue, Irvine, CA, 92617, USA*
[2] *Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY, 13210, USA*

(Received: 16 June 2014; Accepted: 3 October 2014)

In this paper we explore the tradeoff between the leakage power and fan power to dynamically migrate tasks to minimize the overall power consumption in a homogeneous many-core processor. Our analysis shows that the overall power can be minimized if a task allocation for minimum peak temperature is adopted together with an intelligent fan speed adjustment that finds the optimal tradeoff between fan power and leakage power. We propose a method to compute the lower bound on the minimum peak temperature among all possible allocations of given a task set. We further propose two global heuristic task mapping algorithms and a multi-agent distributed task migration framework that minimizes the peak temperature during runtime. The proposed framework achieves large fan power saving as well as overall power reduction. Experimental results show that, given a tight temperature constraint, our distributed task migration policy can save up to 38.5% fan power and 28.9% overall system power compared to the best random mapping policy. Our data also show that the overall system power is insensitive to the task allocation when the temperature constraint is loose.

Keywords: Low Power, Task Allocation, Thermal Aware, Multi-Agent Distributed Framework.

1. INTRODUCTION

The ever-increasing power consumption of the computing device challenges the cooling system at different levels. At data center level, the cooling infrastructure is becoming a limiting factor. The annual cooling cost for a large data center can reach up to tens of millions of dollars. At micro-architecture level, increased power density has set up a "Power Wall" which blocks the microprocessor's performance improvement, and the clock frequency growth is restricted due to the thermal issue. To cool down the processor, a typical cooling fan can consume up to 20%∼50% power budget of a server.[3, 12, 26]

Multi-core architecture has recently become the dominant design platform as it explores task and application parallelism in a power efficient way and hence relieves the power and thermal crisis. With the unprecedented number of transistors integrated on a single chip, the current multi-core trend may soon progress to hundreds or thousands of cores era. Examples of such system are the 80 tile network-on-chip that has been fabricated and tested by Intel,[20] Tilera's 64 core TILE64 processor,[1] and Intel's Single Chip Cloud Computer (SCC).[10]

Even in a homogeneous multi-core system, highly heterogeneous workload on different cores can produce local hotspot and create large thermal gradient. Elevated core temperature increases leakage current and stresses the cooling system. The cooling fan has to operate at a speed to accommodate the worst case power density and guarantee the chip temperature under a safe threshold anywhere and anytime. This would require the fan operating at high speed to maintain fast air flow and strong heat dissipation ability. However, operating at high speed for long time consume more energy and reduce fan life time.[3]

This work is an extension of our original research.[22] In this paper, we focus on the overall power consumption of a homogeneous multi-/many core processor. This consists of three components, dynamic power, static power, and fan power. The goal is to find the efficient task mapping that minimizes the overall power. Due to the homogeneity, task mapping has little impact on the total dynamic power consumption. However, it changes

*Author to whom correspondence should be addressed.
Email: yangge@broadcom.com

the temperature distribution across the system and can potentially affect the leakage power and fan power. While the leakage power is determined by the average temperature, the fan power is determined by the peak temperature. Intuitively they require different optimization techniques. However, for a given workload, the chip leakage power can be approximated to a linear function of the convective resistance of the cooling system while the fan power is an inverse cubic function of the same parameter. As we will show in Section 4 that the impact on leakage power from task mapping is negligible if the fan speed is given. Our analysis shows that the overall power can be minimized if a task allocation for minimum peak temperature is adopted together with an intelligent fan speed adjustment that finds the best tradeoff between fan power and leakage power.[17] We also find that the impact of task allocation on the overall system power is significant when the temperature constraint is tight. When the temperature constraint is loose, the overall system power is insensitive to task allocation.

We formulate the minimum peak temperature task allocation problem as a zero-one linear programming and study the lower bound of its solution. Two centralized heuristic algorithms with linear complexity are proposed, which produce peak temperature close to the lower bound. In reality, the overhead for centralized monitor and control will be prohibitively large when the number of cores increases. Therefore, we further propose an agent based distributed task migration algorithm for peak temperature reduction. Our agent based algorithm has good scalability and achieves up to 28.9% power savings compare to a random mapping policy.

The following summarizes the key contributions of this work.

- We provide an algorithm that finds the lower bound of the peak temperature for a given task set over all possible task mappings. This consequently defines the lower bound of the cooling fan power. The derivation procedure also suggests an "ideal" (not necessarily feasible) task mapping to reach the lower bound. Following the "ideal" mapping, two low overhead task mapping heuristics, which achieve near optimal peak temperature, are proposed.
- We also propose a distributed task mapping algorithm that is equally powerful as the centralized approach in peak temperature reduction. The disturbed algorithm does not require any central control, so the communication cost and task migration overhead for each core remain constant as the number of cores in the system increases.

Comparing to our original work in Ref. [22], the first major extension of this paper is a thorough study of the problem of peak temperature minimization with fixed task set. The lower bound of peak temperature is derived by relaxing the original binary linear programming problem to a standard convex linear programming problem. A constructive technique is proposed that leads to the optimal task allocation for minimum peak temperature. Two task mapping heuristics are further presented. Both of them convert the original problem to a Linear Sum Assignment Problem (LSAP).

The second major extension of this paper is the enriched experimental results section. We compare the lower bound of the peak temperature with the actual peak temperature obtained using centralized task allocation, distributed task allocation and random allocation. Our experimental results show that distributed policy achieves similar peak temperature reduction as centralized policies. We also discuss how to improve the distributed policy for extreme cases in which all high power tasks are initially mapped close to each other.

The rest of the paper is organized as follows: Section 2 reviews the previous work. Section 3 introduces the multi-/many-core system model, the system power, thermal model and cooling system model. We formulate the task allocation problem in Section 4. In Section 5 we present the global and distributed temperature aware task migration algorithm. Experimental results are reported in Section 6. Finally, we conclude the paper in Section 7.

2. RELATED WORK

Various dynamic thermal management (DTM) techniques have been studied at different levels.[6, 8, 14] Most of these works rely on a widely used thermal modeling tool Hotspot[18] for fast thermal analysis. At micro-architecture level, DTM techniques such as clock gating, dynamic voltage and frequency scaling (DVFS), thread migration has been thoroughly explored.[8]

At system level, different approaches have been taken to tackle the issues of high processor operating temperature. Thermal aware task allocation and task migration has been studied in Refs. [6,15]. A multiple-input-multiple-output optimal control theory based power and thermal control algorithm has been proposed in Ref. [21]. The algorithm can control the power of the chip to a specific set point and maintain the chip temperature under a threshold. All these works use a centralized controller to monitor and manage system dynamics. Centralized control does not scale very well as the number of processors increases, because the complexity of the optimization problem and the communication overhead could grow exponentially. Several proactive thermal management scheme has been proposed.[7, 23] They utilized different temperature prediction model to accurately estimate the future temperature in different scenarios and take actions in advance to prevent thermal emergencies. However, these works only focus on temperature optimization without considering its impact on cooling fan power.

The problem of reducing peak power and temperature for many-core chips has also been addressed in some recent works.[24, 25] Ref. [24] proposes a space/timing-division multiplexing based technique to reduce the peak

power consumption of a 3D many-core chip. The authors tackle the peak power reduction problem using two steps. In the first step, they determine the minimum number of power converters needed to satisfy the power demands of all cores and classify converters and cores into different subgroups based on their voltage level. In the second step, workload is further balanced among cores within a subgroup and across timing slots. Ref. [25] proposes a three-level thermal aware task scheduler to reduce the NoC temperature. The global level scheduler determines the priority and dependency among tasks. The cluster level scheduler performs thermal aware task mapping. And the local scheduler executes the tasks based on their priorities. While Ref. [24] aims at peak power reduction, our goal is to achieve lower peak temperature, which helps to reduce the fan power and overall system power. This work also differs from Ref. [25] in several aspects. First, it takes advantage of the variation in the heat dissipation ability of cores during task mapping. Second, our task mapping algorithm is truly distributed, whereas Ref. [25] still needs a central scheduler.

At higher level, power and temperature management techniques for servers, for ensembles and even for data centers have been proposed in the previous works. In Ref. [14], a model for data center air conditioner cooling efficiency has been proposed. In Ref. [19], the heat transfer in data center has been studied thoroughly and a linear heat recirculation model has been introduced. Based on these works, an online workload allocation algorithm for data center has been proposed in Ref. [16]. They predict the future incoming requests and solve an integer linear programming problem online to allocate the workload and optimally turn on or turn off those servers in a data center.

Recently, the cooling fan power optimization has received noticeable attention. In Ref. [17], a joint fan power and processor leakage power consumption optimization has been considered. They formulate the problem as a convex optimization problem and solve it to obtain the optimum fan speed. Yet this work assumes fixed task mapping, while our work exploiting the benefits of task mapping on temperature reduction. In Ref. [3], fan cooling cost minimization for a multi-machine system has been studied. The authors intelligently adjust the workload at virtual machine level and CPU socket level to achieve large fan energy savings.

3. SYSTEM MODEL

3.1. Processor Model

In this paper, we consider a tile-based network-on-chip many-core architecture.[20] Each tile is a processor with dedicated memory and an embedded router. It will also be referred to as *core* in this paper. All the processors and routers are connected by an on-chip network where information is communicated via packet transmission. We refer to the cores that can reach to each other via one-hop communication as the *nearest neighbors*.

We assume the existence of a temperature sensor on each core. A temperature sensor can be a simple diode with reasonably fast and accurate response.[8] We also assume that a dedicated OS layer is running on each core that provides functions for scheduling, resource management as well as communication with other cores.

3.2. Processor Thermal Model

Due to the duality between heat transfer and RC circuits, we abstract the many-core system as an RC network. Let n denote the number of all thermal nodes in the system, including those in the heat sink layer and heat spread layer. Let N denote the number of processors in the system. The relation between n and N is determined by the equation $n = 4 \times N + 12$.[18] Let TSS_i and P_i denote the steady state temperature and average power consumption of node i. Let TSS and P denote vector of TSS_i and P_i, $1 \le i \le n$. When the system reaches the steady state, the temperature of each thermal node is a linear function of power consumptions $P_1, P_2, \ldots, P_n$. The relation can be represented by the following equation

$$TSS = G^{-1}P \tag{1}$$

where $G^{-1} = [g_{ij}]$ is the inverse matrix of thermal conductance matrix G. We simplify Eq. (1) by keeping only the thermal nodes related to the processors:

$$\begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} \tag{2}$$

where N is the number of processors, and $D_i = \sum_{j=N+1}^{n} g_{ij} \cdot P_j$. In this equation, g_{ij} are thermal RC network related parameters. They depend on the chip physical layout, thermal parameters of the material, etc. They are constants for a given chip. P_j, $j \in [N+1, n]$ are the power consumption of the thermal nodes correspond to heat sink and heat spreader and they does not change. Please note that heat sink and heat spreader does not really consume power, but Hotspot thermal modeling tool converts the ambient temperature to the equivalent power consumption of these nodes. The coefficients g_{ij} and D_i, $1 \le i, j \le N$ can be obtained by offline analysis. Equation (2) shows that the steady state temperature of each processor is a linear function of the average power consumptions on all processors and increasing/decreasing the power consumption of one processor will have an impact on the steady state temperature of all other processors.

3.3. Processor Cooling Model

Using TILERA TILE64 processor[1] and Intel SCC[10] as reference, we assume that there is only one standard heat

sink and one cooling fan for the entire many-core system. Our cooling system model follows the previous works in Refs. [3 and 17].

The heat generated in the die layer is transferred through the heat sink layer to the ambient environment and is brought away by the cool air flow provided by the fan. The speed of the air flow determines how efficiently the heat can be dissipated and thus determines the temperature of the die. This heat dissipation ability is characterized by a convective thermal resistance R_{conv}. The faster of the air flow speed, the more easily the heat can be dissipated, and the lower the convective resistance will be. According to Ref. [3] the convective resistance R_{conv} is proportional to $V^{-\alpha}$, where V is the airflow speed and α is a constant between 0.8 and 1.0. The airflow speed is determined by the fan speed, which in turn controls the fan power consumption. It has been pointed out in Ref. [17] that the fan power has cubic relation with the fan speed, i.e., $P_{\text{fan}} \propto V^3$. With all this information, we obtained the relation between the fan power consumption P_{fan} and convective thermal resistance R_{conv}.

$$P_{\text{fan}} \propto R_{\text{conv}}^{-\frac{3}{\alpha}} \tag{3}$$

We next model the relation between the convective resistance and the die temperature. Although the Hotspot[18] provides a detailed and accurate thermal model at micro-architecture level, its complexity is too high to be used analytically. And it does not directly reveal the relation between convective resistance and the die temperature. Therefore, we adopted a simple yet accurate model[3] as shown in Figure 1. In this model, P_i, C_i, and R_i are the power consumption, thermal capacitance and die to package thermal resistance of processor i respectively. R_{hs} is the thermal resistance of heat spreader and heat sink, and R_{conv} is the convective resistance.

Similar to Refs. [3 and 17], we are only interested in the temperature at steady state when the system reaches the equilibrium. This is because the time constant of heat sink is much larger than the time constant of the core. Therefore all the capacitors in the system are open circuit and only thermal resistances will be considered. Then the die temperature T_i of core i can be computed as

$$T_i = P_i R_i + R_{\text{hs}} \sum_{i=1}^{N} P_i + R_{\text{conv}} \sum_{i=1}^{N} P_i \tag{4}$$

If the power consumption of core i does not change, the die temperature of core i is a linear function of R_{conv}. To verify the simple model, we run the simulation in Hotspot to obtain the die temperature of core by varying the convective thermal resistance. Figure 2 shows that the simulated core temperature and the core temperature predicted by the linear model matches very well.

3.4. Leakage Power Model

The leakage power consumption of a processor depends on the die temperature, supply voltage and a number of other factors. If the supply voltage is constant, the leakage power consumption can be expressed as follows:[17]

$$P_{\text{leak}} = A_1 T_d^2 e^{A_2/T_d} + A_3 \tag{5}$$

Where A_1, A_2 and A_3 are constants that depend on processing technology and supply voltage, and T_d is the die temperature. It has been pointed out in Ref. [13] that the leakage power can be approximated using a linear model with less than 5% error over a large temperature range from 20 °C to 120 °C. We approximate the leakage power using its first order Taylor expansion at 80 °C and compare the linear approximation model with the original model in Figure 3. The green line is the linear approximation while the red line is the original model given by Eq. (5). Figure 3 shows that the linear model has very small error compared to the original model over the range of normal operating temperature, which is between 60 °C and 100 °C.

Based on the above results, we approximate the leakage power of the ith core using a linear model

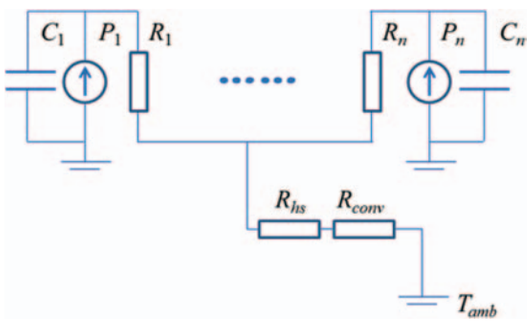

Fig. 1. Simplified multiprocessor thermal model.

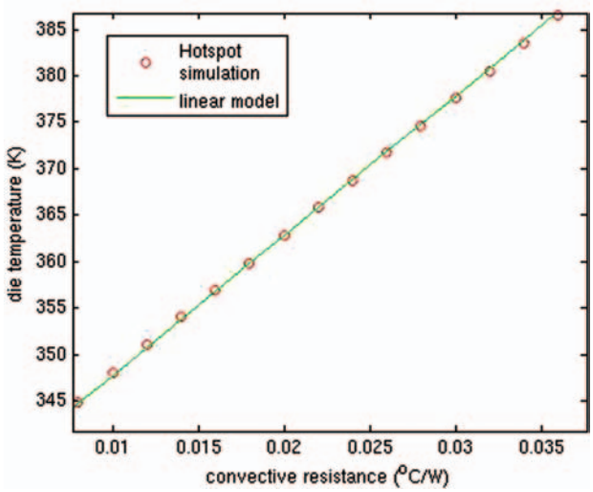

Fig. 2. Linear approximation of relation between die temperature and convective resistance.

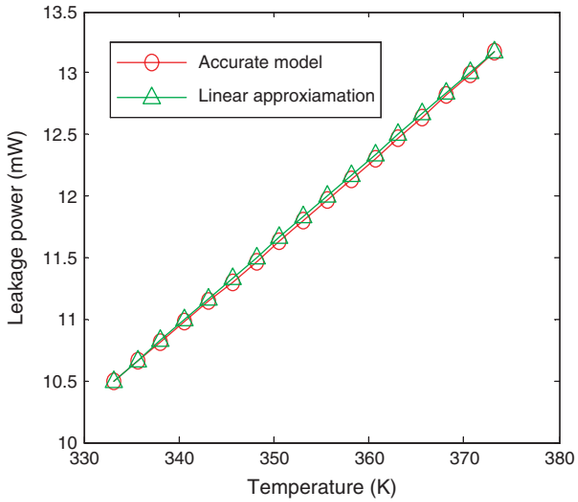

Fig. 3. Linear approximation of leakage power model.

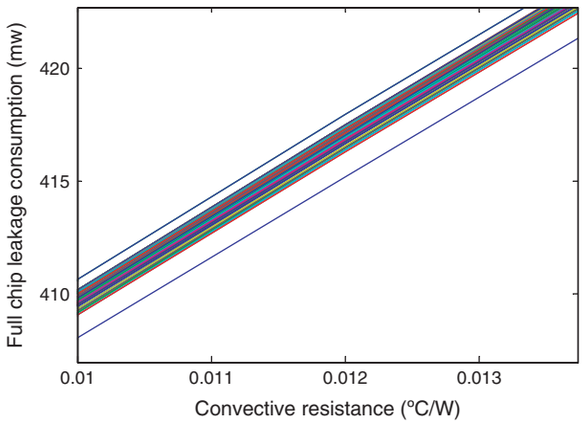

Fig. 4. The relation between full chip leakage power consumption and different task allocations.

$P_{\text{leak},i} = aT_{d,i} + b$, where $T_{d,i}$ is the average die temperature of the core, a and b are two scalars. The total leakage power consumption can be simply calculated as $P_{\text{leaktotal}} = a\sum T_{d,i} + Nb = NaT_{d-avg} + Nb$, where $T_{d-avg} = \sum_{i=1}^{N} T_{d,i}/N$ is the average temperature of N cores. Thus the total chip leakage power can be approximated as a linear function of average die temperature. Because $T_{d,i}$ is linearly proportional to R_{conv}, the leakage power $P_{\text{leaktotal}}$ is also a linear function of the convective resistance.

4. PROBLEM FORMULATION AND ANALYSIS

In this section, we study the impact of task allocation on the overall system power. The overall power consumption is the sum of the CPU power consumption and the fan power consumption while the CPU power consumption consists of dynamic power and leakage power. Therefore the overall power consumption model can be written as follows.

$$P_{\text{total}} = P_{\text{dyn}} + P_{\text{leak}} + P_{\text{fan}} \tag{6}$$

In a homogeneous multi-core system, task allocation has little impact on the dynamic power consumption because all cores are identical. However, because task allocation changes the temperature distribution across the system, it has the potential to change the leakage power and fan power, which are temperature related.

To show the relation between task allocation and leakage power consumption, we randomly generate 100 groups of task allocation for a given workload and compare their leakage power consumption on a 36-core multiprocessor. The workload consists of 36 tasks whose power consumption varies from 10 mW to 20 mW (details about workload generation are described in Section 6). Figure 4 shows the leakage power for all 100 groups as the convective resistance increases. The leakage power consumption for the worst mapping and the best mapping differs only by less than 1% for a given convective resistance. This is intuitively correct. The leakage power is linearly proportional to the average die temperature, which is determined by the average power density across the chip. Since the task allocation has little impact on the processor's dynamic power consumption, which is still the dominant part of the CPU power consumption when it is actively running, it does not significantly change the average chip temperature either. Consequently, the leakage power remains stable. Figure 5(a) shows the average die temperature for those 100 different random mappings as the convective resistance increases. (The blue line that lies at the bottom corresponds to the task allocation that is found by our multi-agent distributed task migration framework that will be introduced in the next section.) As we can see, the maximum difference in average die temperature is less than 1 °C.

From the experimental results we have two observations, (1) for a given R_{conv}, the leakage power can be considered to be independent of the task allocation, (2) when the workload is given, the only parameter that controls the leakage power is the fan speed which is reflected by R_{conv}. Their relation can be represented by a linear function: $P_{\text{leak}} = c_1 R_{\text{conv}} + c_2$.

On the other hand, different task allocations significantly affect the peak temperature. In order to bring the peak temperature below the constraint, the fan speed needs to be adjusted accordingly, which in turn leads to different R_{conv}. For example, Figure 5(b) shows the maximum chip temperature of 100 different mappings as the convective resistance increases. (Again, the blue line that lies at the bottom corresponds to the task allocation that is found by our multi-agent distributed task migration framework.) We can see that the difference in peak temperature is more than 10 °C. Note that because the average temperatures for different allocations are almost the same, the task allocation that gives the lowest peak temperature is the one

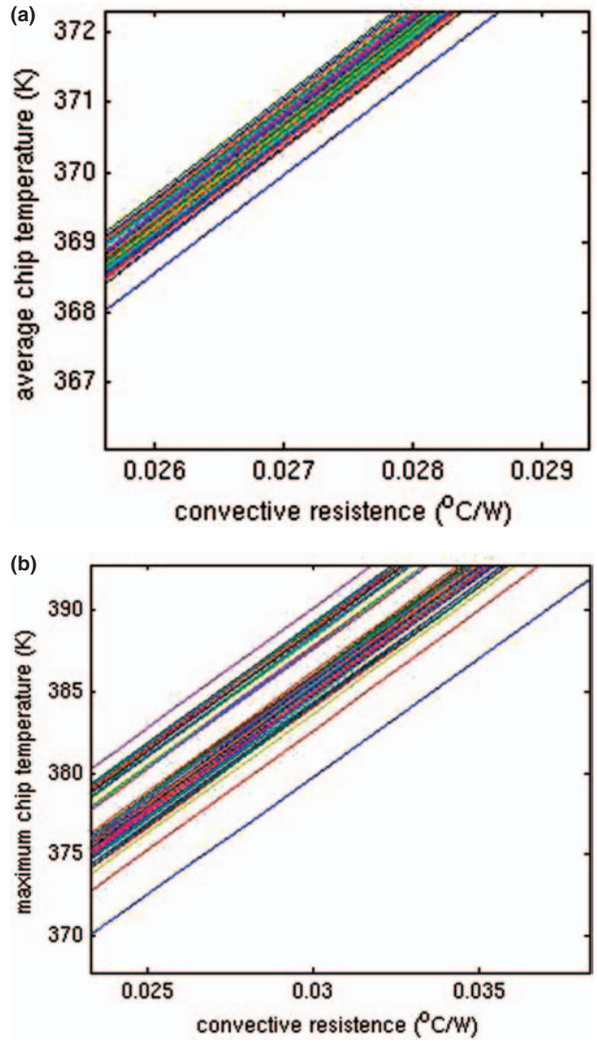

Fig. 5. Comparison of average and maximum temperature of different task allocations.

that generates the most balanced temperature distribution. A task allocation that generates highly unbalanced temperature distribution will put more stress on the cooling fan, thus increase the fan power consumption. However, as the speed of cooling fan increases, the average chip temperature will decrease and therefore bring down the leakage power. When searching for the optimal task mapping, we need to consider the tradeoff between fan power and leakage power.

Because P_{dyn} is independent of thermal convective resistance, P_{leak} is linearly proportional to convective resistance and P_{fan} is an inverse cubic function of the convective resistance, the overall power consumption is a convex function of the convective resistance. There will be an optimal convective resistance R^*_{conv} (corresponding to the optimal fan speed) which minimizes the overall system power. Furthermore, because task allocation has little impact on P_{dyn} and P_{leak} for a given workload, the same relation between the overall power consumption and the convective resistance can be applied to different task allocations.

Figure 6 shows the overall power consumption and the peak temperature under different task allocations as functions of the convective resistance. Figure 6(a) shows the scenario when the temperature constraint is strict and the convective resistance (i.e., $r_{1,\max}$ and $r_{2,\max}$) that exactly bring the peak temperature to the constraint are located to the left of R^*_{conv}. In this case the overall power is dominated by the fan power. Increasing the fan speed can only increase the overall power consumption. The best task allocation that minimizes the overall system power is allocation 2 which has lower peak temperature than allocation 1 under the same R_{conv}. Figure 6(b) shows the scenario when the temperature constraint is loose and the convective resistance that could bring the peak temperature to the constraint are located to the right of R^*_{conv}. With a loose temperature constraint, the fan power does not dominate the overall power alone; leakage power plays an important role as well. Increase the fan speed will increase the fan power but also reduce the temperature and leakage power. In this case, the leakage power reduction surpasses the fan power increase. For both allocation 1 and 2, setting the convective resistance to R^*_{conv} can minimize the overall power while satisfying the temperature constraint. In this scenario, power consumption is not sensitive to task allocation. Any allocation scheme whose r_{max} is greater than R^*_{conv} could be used together with a fan speed adjustment method to find the optimal tradeoff between the fan power and the leakage power. Obviously, among all possible task allocations, the allocation that minimizes the peak temperature is most likely to satisfy this property.

Based on these observations, we concluded that, to optimize the overall power consumption, there are two steps. First is to find the task allocation that minimizes the peak temperature. Second is to adjust the fan speed to find the optimal tradeoff between fan power and leakage power such that the overall power consumption is minimized and the temperature constraint is satisfied. The second step could be achieved by using feedback control or the method proposed in Ref. [17] while the first step will be discussed in detail in the following sections. Please note that in the first step, we fix the fan speed/convective resistance and use the temperature model in Section 3.2 while in the second step, we fix the task mapping and use the cooling model in Section 3.3. Our task allocation algorithm can also be combined with other active cooling methods, e.g., micro-fluidic cooling. Micro-fluidic cooling pumps the coolant into the micro-channels. Its cooling ability as well as the pumping power depends on the fluid flow rate.[28] This is conceptually similar to the relation between cooling fan's power consumption and the fan speed.

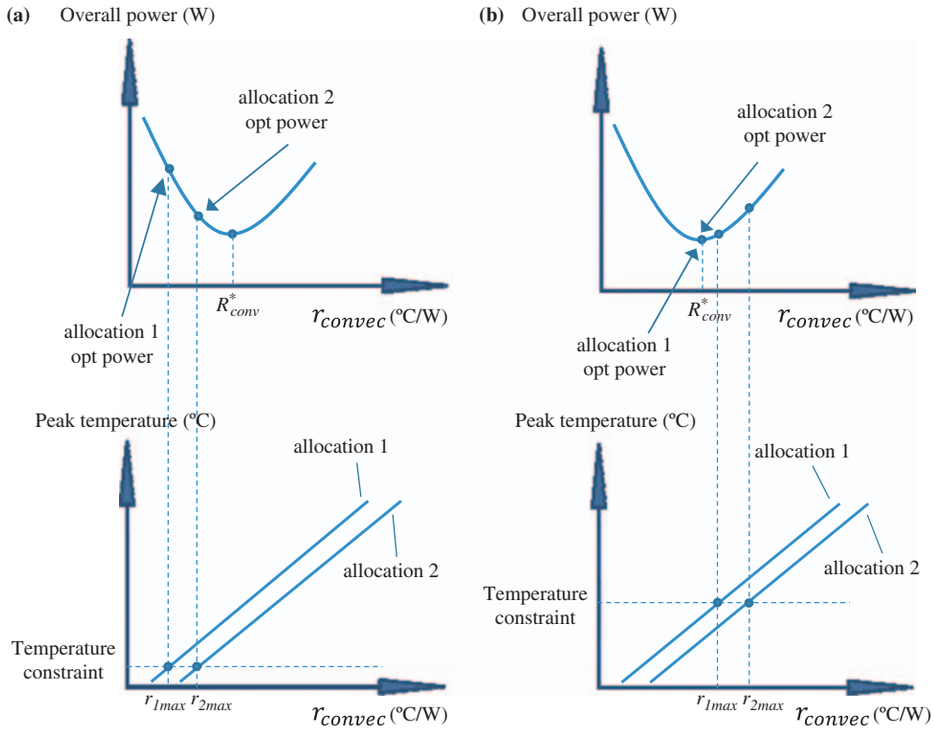

Fig. 6. The overall power consumption depend on convective resistance.

5. TASK ALLOCATION FOR MINIMAL PEAK TEMPERATURE

Based on the analysis in the previous section, we will focus on searching for the optimal task allocation that minimizes the peak temperature among all cores.

In the following subsections, we first present an exact formulation of this problem, which is a zero-one min-max problem. Because the exact solution of the zero-one min-max problem is hard to find, we do not try to solve it directly. Instead, we will prove a sufficient condition for the optimal task allocation. Based on the sufficient condition, we derive a lower bound for the optimization problem. We then present two heuristics for centralized management of task allocation. At the end, we present a distributed multi-agent task migration framework that searches for the best task allocation during runtime.

5.1. An Exact Formulation

Given a floorplan of a multi-processor system with N cores integrated on a chip, we assume that the thermal conductance matrix can be characterized by offline training. We further assume that the given workload consist of N different tasks $\{\tau_1, \tau_2, \ldots, \tau_N\}$ whose power consumption $\{P_1, P_2, \ldots, P_N\}$ can be obtained through offline training or online estimation by observing the event counters. We assume that the power consumption is a constant for each task, because we are only concerned about the steady state temperature. If a processor runs more than one task, we use the average power of the task set. On the other hand, if the number of tasks is less than the number of cores, we simply add some dummy tasks with zero power consumption.

Our goal is to obtain a mapping between the N tasks and the processors such that the resulting maximum temperature among all the cores is minimized. For each task k and processor j, there is a variable x_{jk}. Variable x_{jk} is 1 when task k is mapped to processor j, otherwise it is 0. We formulate the problem as a zero-one min-max linear programming as follows:

$$\min_i \max_i \left(\sum_{j=1}^{N} \sum_{k=1}^{N} g_{ij} x_{jk} P_k \right) + D_i \tag{7}$$

$$\text{Subject to: } \sum_{j=1}^{N} x_{jk} = 1, \quad \forall k = 1, \ldots, N \tag{8}$$

$$\sum_{k=1}^{N} x_{jk} = 1, \quad \forall j = 1, \ldots, N \tag{9}$$

$$x_{jk} \in \{0, 1\} \tag{10}$$

Constraint (8) guarantees that a processor is only occupied by one task and constraint (9) ensures that a task can only be mapped to one processor. The item within the min-max operator in the objective function is the temperature

of the ith core. To see this, we rewrite the Eq. (2) as follows:

$$\begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NN} \end{pmatrix} \times \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} \quad (11)$$

where $X = [x_{ij}]$ is a permutation matrix which assigns the N tasks to the processors. Expanding the right hand side of the equation, we can have $T_i = \sum_{j=1}^{N} \sum_{k=1}^{N} g_{ij} x_{jk} P_k$. Then the objective function min-max(T_i) is to minimize the maximum temperature among all N processors.

By simple transformation, the min-max problem can be converted to traditional linear programming:

$$\min \quad u \quad (12)$$

$$\text{Subject to:} \quad \left(\sum_{j=1}^{N} \sum_{k=1}^{N} g_{ij} x_{jk} P_k \right) + D_i \le u, \quad \forall i = 1, \ldots, N \quad (13)$$

$$\sum_{j=1}^{N} x_{jk} = 1, \quad \forall k = 1, \ldots, N \quad (14)$$

$$\sum_{k=1}^{N} x_{jk} = 1, \quad \forall j = 1, \ldots, N \quad (15)$$

$$x_{jk} \in \{0, 1\} \quad (16)$$

Solving the above zero-one min-max linear programming is very time consuming. For example, for a problem with 36 cores there will be 1296 binary variables, it would take more than two days to solve this problem using the open source linear programming solver lp_solver[2] on a 3.2 GHz Quad core Xeon processor. It cannot be used for online power and thermal optimization in large size many-core systems where the core counts could go up to hundreds and thousands.[4]

5.2. A Sufficient Condition for Optimum Task Allocation

From the analysis in Section 4, we observe that the average chip temperature is hardly affected by different task allocations but the peak temperature of the chip could have significant difference. This observation implies that the minimum peak temperature task allocation is the one that makes the temperatures evenly distributed. Therefore we have the following proposition:

PROPOSITION 1. *If there is one task allocation such that the core temperatures are the same for all cores on the chip, then this allocation is an optimum solution for the optimization problem* (12)∼(16).

In order to prove the above proposition, we rewrite the linear relation between temperature and power consumption in Eq. (2) as following

$$\begin{pmatrix} b_{11} & \cdots & b_{1N} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NN} \end{pmatrix} \begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} + \begin{pmatrix} C_1 \\ \vdots \\ C_N \end{pmatrix} = \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} \quad (17)$$

The thermal conductance matrix $\mathbf{B} = [b_{ij}]$ has some special properties. Firstly, it is a sparse matrix. An entry b_{ij} in the matrix is non-zero if and only if core i and core j are physically adjacent or $i = j$. Secondly, the diagonal elements in the matrix $\mathbf{B}$ are positive. Thirdly, the matrix $\mathbf{B}$ is a strictly diagonally dominant matrix, which means $b_{ii} > \sum_{j \neq i} |b_{ji}|$. These three properties of the thermal conductance matrix are guaranteed by the Hotspot thermal modeling tool.[18]

We prove the Proposition 1 by using contradiction. Assume $\mathbf{P}^* = (P_1^*, \ldots, P_N^*)$ is a task allocation such that all cores have the same temperatures, i.e., $\mathbf{T}^* = (T_1^*, \ldots, T_N^*)$, where $T_1^* = \cdots = T_N^*$. We further assume that there is a task allocation $\mathbf{P}' = (P_1', \ldots, P_N')$ such that the core temperature vector $\mathbf{T}' = (T_1', \ldots, T_N')$ are strictly smaller than $\mathbf{T}^*$, i.e., $T_i' < T_i^*, \ \forall i = 1, \ldots, N$. Because $\mathbf{P}^*$, $\mathbf{T}^*$ and $\mathbf{P}'$, $\mathbf{T}'$ are satisfying Eq. (17), we insert them into the equation and subtract one from the other and get the following equation

$$\begin{pmatrix} b_{11} & \cdots & b_{1N} \\ \vdots & \ddots & \vdots \\ b_{N1} & \cdots & b_{NN} \end{pmatrix} \begin{pmatrix} \Delta T_1 \\ \vdots \\ \Delta T_N \end{pmatrix} = \begin{pmatrix} \Delta P_1 \\ \vdots \\ \Delta P_N \end{pmatrix} \quad (18)$$

Where $\Delta T_i = T_i^* - T_i' > 0$. We multiply both sides of the equation by the vector $\mathbf{I} = (1, 1, \ldots, 1)$ and obtain

$$(B_1, \ldots B_N) \begin{pmatrix} \Delta T_1 \\ \vdots \\ \Delta T_N \end{pmatrix} = \sum_{i=1}^{N} \Delta P_i = 0 \quad (19)$$

Where $B_i = \sum_{i=1}^{N} b_{ji}$. Because $\mathbf{B}$ is strictly diagonally dominant, $B_i > 0, \ \forall i = 1, \ldots, N$, and because$\Delta T_i > 0, \ \forall i = 1, \ldots, N$, the left side of (19) is strictly greater than 0 and the equation cannot hold. Therefore the task allocation $\mathbf{P}' = (P_1', \ldots, P_N')$ does not exist, which means the task allocation $\mathbf{P}^*$ is the optimum task allocation which minimizes the peak temperature.

From this proposition, we immediately have the following corollary.

COROLLARY 2. *There is no two task allocations* $\mathbf{P}'$ *and* $\mathbf{P}''$, *such that* $\mathbf{T}'$ *is completely cooler than* $\mathbf{T}''$, *i.e.,* $\Delta T_i = T_i' - T_i'' < 0, \ \forall i = 1, \ldots, N$.

The proof of the corollary can be obtained in the exactly the same way as Proposition 1.

5.3. A Lower Bound for the Optimum Task Allocation Problem

Proposition 1 is only a sufficient condition for the optimality but not a necessary condition. In fact, there might not be a task allocation such that the temperatures of all cores are exactly the same. In this subsection, we consider a relaxed Linear Programming (LP) and derive a lower bound for the original task allocation problem.

In the original problem, P_i cannot be selected arbitrarily and could only be the power of one task. Now, we relax this constraint, allowing P_i to be any real number between 0 and P. And we formulate the following LP problem.

$$\min \quad u \tag{20}$$

$$\text{Subject to:} \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} = \begin{pmatrix} u \\ \vdots \\ u \end{pmatrix} \tag{21}$$

$$P_i \geq 0, \quad \forall i = 1, \ldots, N \tag{22}$$

$$\sum_{i=1}^{N} P_i = P \tag{23}$$

P is the total power consumption of all tasks. If there is a task allocation that satisfies the above LP problem, then according to Proposition 1 it gives the minimum peak temperature. It also gives the lower bound of the original task allocation problem.

The above LP program (20)∼(23) can be readily solved by a LP solver because the object function and the constraints are all affine functions. The invertibility of the matrix $\mathbf{B} = [b_{ij}] = \mathbf{G^{-1}} = [g_{ij}]^{-1}$ makes the LP problem analytically solvable. The optimum temperature can be expressed in the following equation

$$T_i^* = \frac{P - \sum_{i=1}^{N} C_i}{\sum_{i,j} b_{ij}}, \quad \forall i \in [1, N] \tag{24}$$

And the optimum power distribution can immediately obtained by (17).

5.4. A Heuristic Task Allocation Algorithm for Average Temperature Minimization

In this section, we present a heuristic algorithm to find an approximate task allocation that gives the result close to the lower bound. We first transform the objective function (20) to the following format.

$$\min u = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_{j=1}^{N} g_{ij} P_j + D_i \right) \tag{25}$$

Because D_i and N are constants, they could be dropped in (25). The new objective function is reduced to:

$$\min \sum_{i=1}^{N} \left(\sum_{j=1}^{N} g_{ij} P_j \right) \tag{26}$$

We transform the objective function in (26) by switching the order of summation as following

$$\sum_{i=1}^{N} \left(\sum_{j=1}^{N} g_{ij} P_j \right) = \sum_{j=1}^{N} \left(\sum_{i=1}^{N} g_{ij} P_j \right) = \sum_{j=1}^{N} P_j \left(\sum_{i=1}^{N} g_{ij} \right) = \sum_{j=1}^{N} P_j G_j \tag{27}$$

where $G_j = \sum_{i=1}^{N} g_{ij}$ is a set of constants. As we can see, for arbitrary k and l in $[1, \ldots, N]$. If $G_l \leq G_k$ and $P_l \leq P_k$, then

$$(G_l - G_k)(P_l - P_k) \geq 0 \tag{28}$$

Thus

$$G_l P_l + G_k P_k \geq G_l P_k + G_k P_l \tag{29}$$

Equation (30) indicates that, to reduce $\sum_{j=1}^{N} P_j G_j$, we should map task k (which has greater power consumption) to core l (which has lower thermal resistance), and task l with core k.

Therefore, we further sort the tasks according to the descending order of their power consumptions and sort the G_j in based on the ascending order of their value, i.e., $\mathbf{P} = (P_1, \ldots, P_N)$ such that $P_1 \geq \cdots \geq P_N$ and $\mathbf{G} = (G_1, \ldots, G_N)$ such that $G_1 \leq \ldots \leq G_N$. The tasks are mapped to cores accordingly.

This is actually a Linear Sum Assignment Problem (LSAP). Therefore, we name the first heuristic algorithm as LSAP-1.

5.5. A Heuristic Task Allocation Algorithm Based on Optimum Power Distribution

In this section, we present the second heuristic task allocation algorithm. The LP specified in (21)∼(23) gives a lower bound of peak temperature for a given workload. It also provides an ideal task allocation, which may serve as a reference. The goal of our second heuristic algorithm is to find a task allocation which minimizes the difference between the current workload distribution and the optimal workload distribution.

Let us denote the reference workload distribution obtained by Section 5.3 as $\mathbf{P}^r = (P_1^r, \ldots, P_N^r)$, and we formulate our optimization problem as following.

$$\min \quad \sum_{i=1}^{N} (P_i^r - P_i)^2 \tag{30}$$

$$\text{Subject to} \quad \mathbf{P} = (P_1, \ldots, P_N) \text{ is a task allocation} \tag{31}$$

Here, we use the Squared Euclidean Distance to represent the difference between current power distribution and

the optimal power distribution. At the first look, it is a Quadratic Programming (QP) problem. However, similar as the optimization problem in last section, the solution of this problem can be determined analytically without using any quadratic programming solver.

We again start by transforming the objective function. By expanding the Eq. (31), we obtain

$$\sum_{i=1}^{N}(P_i^r - P_i)^2 = \sum_{i=1}^{N} P_i^{r2} - 2 \times P_i^r \times P_i + P_i^2$$
$$= \sum_{i=1}^{N} P_i^{r2} + \sum_{i=1}^{N} P_i^2 - 2\sum_{i=1}^{N} P_i^r P_i \quad (32)$$

Please note that the first two terms in the above equation does not change for any task allocation given the constraint (32). Following the analysis in Section 5.4, we also know that if we arrange the P_i^r and P_i in ascending order, then $\sum_{i=1}^{N} P_i^r P_i$ will be maximized. Again, this problem can be solved by linear sum assignment and we name the algorithm as LSAP-2.

5.6. Distributed Task Migration

Both LSAP-1 and LSAP-2 are centralized approaches. They require a central controller that monitors the temperature and workload distribution of all cores on the entire chip and make global decisions of task allocation. Even though both algorithms have linear complexity, they do not have good scalability when applied to large systems because they require a centralized monitoring and commanding framework across the entire chip. Such framework will suffer from synchronization errors, unpredictable delays and high power consumption as the number of cores scales.[9]

To improve the scalability for multi-/many-core system, in this section, we present a distributed task migration framework that searches the optimal task allocation during runtime.

We denote our multi-agent task migration algorithm as MATM. The framework has a low cost agent residing in each core. It is part of OS based resource management program, which performs thermal-aware task migration. The agent observes the workload and temperature of local processor while communicating and exchanging tasks with its nearest neighbors. The agent based distributed framework has better scalability compared to the centralized method as the communication cost and migration overhead for each core does not increase when the number of cores in the system increases.

The proposed MATM adopts a task exchange based migration scheme. By exchanging tasks, the processors can maintain a balanced temperature distribution and hence reduce the peak temperature.

5.6.1. Communication Protocol

Each core running a MATM agent can be in two phases: execution phase and scheduling phase. These two phases are interleaved. During the execution phase the core executes the current computing task, while during the scheduling interval it initiates task migration request to its nearest neighbor or responds to the task migration request from its nearest neighbor. The scheduling interval can further be divided into four sub-phases: broadcasting self-workload to neighbor cores, receiving workload information from neighbors, sending migration requests to neighbors, exchanging tasks with neighbors. Figure 7 shows the diagram of the communication protocol. We assume that a MPI (Message Passing Interface) based communication is adopted. Therefore two cores do not have to enter the scheduling interval synchronously in order to communicate to each other.

At the beginning of each scheduling interval, an agent on a processor would broadcast its own workload to neighbors and request them sending back their workload. Because the scheduling intervals in all processors are not synchronized, the request is not likely to be checked and responded by neighbor agents right away. On the other hand, because all processors adopt the same execution and scheduling interval, it is guaranteed that all neighboring agents will respond before the next scheduling interval after the request is issued.

After receiving the response of neighbor workloads, the agent performs the MATM algorithm to decide whether to exchange task with neighbors and select which neighbor to exchange task with. Then it will send a migration request to the selected processor. For all other neighboring processors, the agent will also send an acknowledgement to them, which indicates no task exchange. After that, the agent waits for the migration response from the selected processor. Please note that this communication will only be established with its nearest neighbors in a neighborhood

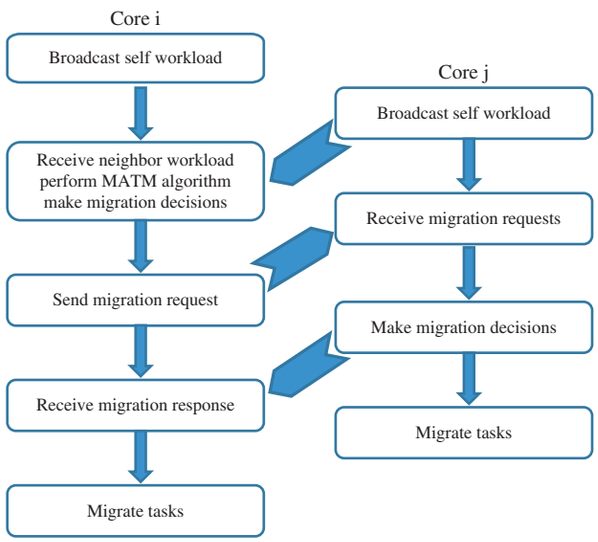

Fig. 7. Diagram of communication protocol.

when a core has workload changes. Therefore the communication overhead is low. Because communication distance is very short (only between nearest neighbors) and the processor does not need to receive the migration response right away, the scheduling interval can be kept very short. We assume that the duration of the scheduling interval is negligible compared to the execution interval and will have little impact on the system thermal characteristics.

5.6.2. MATM Distributed Migration Algorithm

The MATM algorithm can be viewed as a distributed version of the LSAP-1 algorithm proposed in Section 5.4. It distributes the tasks among processors based on their heat dissipation ability. It moves high power tasks to processors with strong heat dissipation capability and low power tasks to processors with weak heat dissipation capability in a neighborhood. By distributing tasks in this way, local hotspots can be mitigated and thus peak temperature of the chip can be reduced.

To determine if an exchange of tasks between two processors is beneficial to the whole system, we consider Eq. (2) again. Assume that core i and j exchange tasks, and their average power consumptions are altered by ΔP_i and ΔP_j respectively. Using Eq. (2), the total die temperature change of all processors in the system after task migration can be calculated as:

$$\sum_{k=1}^{N} \Delta T_k = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j \tag{33}$$

where G_i (or G_j) is the parameter that characterizes the heat dissipation ability of core i (or j) as defined in Section 5.4, i.e., $G_i = \sum_{m=1}^{N} g_{mi}$, $G_j = \sum_{n=1}^{N} g_{nj}$. The temperature contributed by core i running task k can be calculated as $G_i P_k$. If $G_i < G_j$ and $P_i < P_j$, after switching P_i and P_j we will have $\Delta P_i > 0$ and $\Delta P_j < 0$. This leads to $\sum_{k=1}^{N} \Delta T_k <$ in (34). If a task exchange between two neighbor processors leads to $\sum_{k=1}^{N} \Delta T_k < 0$, then this task exchange is beneficial for the system and the task exchange should be carried out.

If an agent found that it is beneficial to exchange task with several neighbor agent, the agent will select a neighbor that leads to maximum temperature reduction, i.e., the minimum $\sum_{k=1}^{N} \Delta T_k$ (because it is negative), and send migration request to the selected neighbor. If an agent received several migration requests from neighbors, it will follow the same criterion to select a neighbor to exchange tasks. We summarized the MATM in Figure 8. Please note task exchange only happens if and only if both $G_i < G_j$ and $P_i < P_j$ conditions are satisfied. So it is impossible for task i and task j to be switched between two cores back and forth continually.

The MATM algorithm is designed to overcome the communication overhead (power, energy and latency) for a many-core system. As pointed out in Refs. [11 and 27], the energy for transferring a data packet from one core to another core is proportional to the number of hops along the path. This is because the switches and buffers in the routers, and the links between routers also consume energy when transferring a packet. For the similar reason, the latency for transferring a packet is also proportional to the number of hops. If we consider the congestions, the energy and latency for communications between two distant cores could be even larger. If communication only happens between neighboring cores, it not only consumes less energy and takes less time, but is also virtually congestion free. Therefore, in a system with NoC based on-chip interconnect, it is preferred to have the communication between neighboring cores than between distant cores.

Algorithm 1 MATM

1. for each neighbor processor j, compute
2. $\Delta T_{ij} = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j$
3. $\Delta T_{\min} = \min(\Delta T_{ij})$
4. Select processor j, and send migration request to it

Fig. 8. MATM algorithm.

6. EXPERIMENTAL RESULTS

We implemented a many-core system simulator using C++. Hotspot[18] is integrated to the simulator to analyze the system thermal behavior. Though the model is scalable for any number of cores, a 36 core system with 6×6 grids is chosen for our experiments due to the limitation of simulation time. Each core has a size of 4 mm $\times$ 4 mm with silicon layer of 24 mm $\times$ 24 mm. All the physical parameters are the Hotspot's default value, except the convective resistance, which is a variable. Task dynamic power is obtained through Wattch power analysis tool. For leakage power, we apply the leakage power model in Ref. [17] and scale it with respect to the dynamic power, so leakage power is about 40% the total processor power.[13] For cooling fan power, we apply the model in Ref. [3] and also scale it with respect to the processor power, so cooling fan power could range from less than 10% to 50% of the overall system power[3, 12, 26] based on the convective resistance.

We carried out experiments using power sequences collected from real applications. We used 9 different CPU benchmarks comprising of 3 SPEC 2000 benchmarks (bzip2, applu and mesa), 4 Mediabench applications (mpeg2enc, mpeg2dec, jpegdec, jpegenc) and 2 telecom applications (crc32 and fft) from MiBench benchmark suite. We collected cycle level power trace by modifying the Wattch power analysis tool.[5] The average dynamic power consumptions and steady state temperatures of each task are summarized in Table I. The workloads of the following experiments are random combinations of multiple copies of these 9 benchmarks. All experiment results reported below are the average of 10 runs.

The experiment is performed on 5 different task sets. Each task set consists of 36 tasks. Each task is random

Table I. Average Power and Steady State Temperature of CPU Benchmarks.

Task ID.	1	2	3	4	5	6	7	8	9
Bench marks	crc32	mp2enc	mp2dec	fft	applu	mesa	bzip2	jpegdec	jpegenc
Avg. power (mW)	24.4	19.4	19	18.5	17.4	17.3	13.3	10.7	10.4
Steady temp. (°C)	99.42	84.17	82.95	81.42	78.07	77.76	65.56	57.63	56.72

selected from the 9 bench marks listed in Table I. We control the selection probability of a benchmark based on its average power consumption so that the average power consumption of the 36 tasks can follow a desired distribution. Figure 9 shows the five different task distribution. The *x*-axis is the task ID in Table I and the *y*-axis is the number of task. Uniform distribution evenly generates tasks with different average power consumptions. Triangular (cool) distribution generates more low power tasks than high power tasks, whereas triangular (hot) distribution generates more high power tasks. Normal distribution generates a set of tasks whose power consumption is mostly clustered around the medium power. On the other hand, inverse normal distribution generates more high power tasks and low power tasks than the medium power tasks.

6.1. Fan Power Savings

Figure 10 shows the r_{max}, i.e., the maximum thermal convective resistance that is required to exactly keep the temperature below 80 °C. Five different task allocation policies are compared, including the allocation that gives the exact lower bound peak temperature (lb), LSAP-1, LSAP-2, MATM, and the best one from 100 groups of random allocation (rand). We also implement a base line policy called base1. Base1 looks the temperature of each core based on initial task mapping. It starts with the hottest core and tries to swap a high power task on a hotter core with a low power task on a cooler core. Among these, all polices need centralized control except MATM. The results show that, to maintain the whole system under the temperature constraint, the minimum fan speed required by MATM based allocation is 15.1% and 9.54% lower than that is required by the rand and base1 policy. The reduced fan speed could bring cubic savings in fan power for the system. And Table II shows the fan power savings of our proposed MATM policy compared to the rand and base1 policy. The MATM can achieve an average of 38.53% and 25.12% fan power savings over rand and base1 policy while maintains the maximum chip temperature under the thermal constraint.

The experimental results show that MATM policy distributes tasks among processors evenly according to a processor's heat dissipation ability. As the result, the temperatures are also distributed evenly across the chip and the maximum temperature is reduced. Hence, the fan can run at a relatively lower speed to meet the temperature constraint and the fan power saving is achieved. Compared to MATM, base1 policy ignores each core's heat dissipation ability, and swaps a high power task with a low power task simply based on core temperature. Although it reduces the temperature of the hotter core, it might as well create new hotspot on the cooler core. Overall, the peak temperature cannot be reduced effectively.

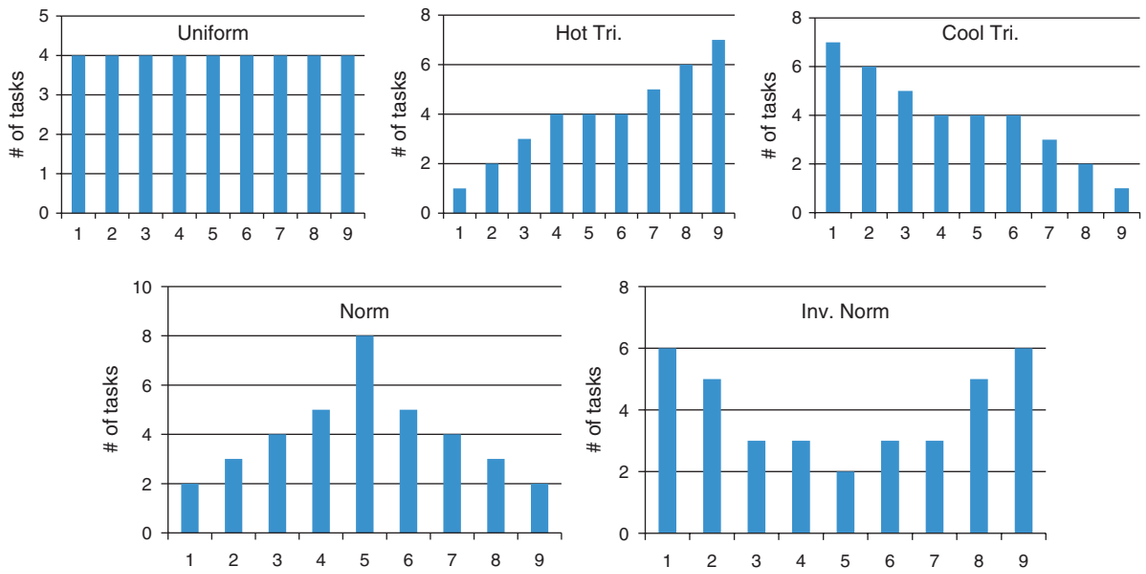

Fig. 9. Different task set generation probability distribution.

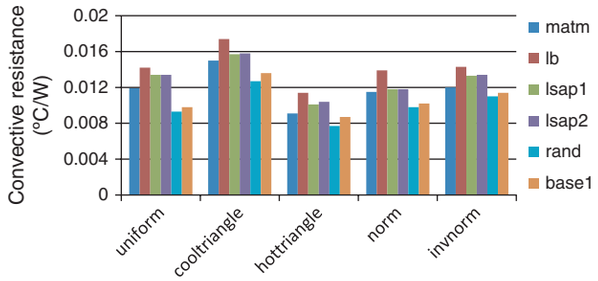

Fig. 10. Convective resistance comparison between 5 different task allocation policies.

The results in Figure 10 also show that, because the MATM is a distributed policy and each processor can only communicate with its nearest neighbor, the algorithm cannot find better task allocation than the two centralized policies LSAP-1 and LSAP-2. Compared to MATM, LSAP-1 could reduce the fan speed by 8.43% and LSAP-2 could reduce the fan speed by 9.94%. The differences of the MATM algorithm and the two global policies are less than 10%. We also observed that although LSAP-1 and LSAP-2 are all global policies, LSAP-2 is constantly better than LSAP-1. This indicates that minimizing the average chip temperature (LSAP-1) is not equivalent to minimizing the peak chip temperature. Finally, the lower bound policy always gives the lowest fan speed. However, it is only less than 15% better than MATM.

6.2. Overall System Power Consumption

In the second experiment, we examine the effect of temperature constraint and task allocation on the overall system power consumption, i.e., the power consumption summation of dynamic power, leakage power and fan power. We select the uniform workload distribution in this experiment. We vary the temperature constraint for 80 °C, 85 °C and 90 °C and compare the power consumption between the five task allocation policies. For all policies, optimal tradeoff point between fan power and leakage power will be searched after the system reaching stable state. As shown in Table III, comparing to best random allocation and base1 policy, MATM based allocation policy could achieve 28.9% and 22.2% overall power savings when the temperature constraint is 80 °C. When the temperature constraint increases to 85 °C and 90 °C, the power saving reduces to 14.8%, 11.8% and 7.2%, 6.0% respectively.

Table II. Fan power savings of MATM compared to the rand and base1 policy.

Workload	Uniform (%)	Cool tri (%)	Hot tri (%)	Norm (%)	Inv norm (%)
Impr. versus rand	52.64	38.72	40.38	36.84	24.06
Impr. versus base1	43.91	25.26	11.54	29.81	15.09

Table III. Overall system power consumption comparison under different temperature constraints.

Overall system power consumption (mW)			
Temp. constraint	80 °C	85 °C	90 °C
rand	2271.04	1543.77	1289.61
base1	2076.01	1491.06	1272.66
MATM	1614.83	1315.59	1196.73

The experimental results show a diminishing power savings as the constraint temperature increases from the Table III, and task allocation gives large power savings especially when temperature constraint is strict. To understand this, we draw the curve of overall power consumption and fan power consumption against the convective resistance curve in Figure 11. When temperature constraint is strict, the convective resistance has to be small to satisfy the constraint. When fan is working in this area, the curve slope is sharp and a little decrease in convective resistance would increase the fan power as well as the overall system power significantly; therefore a better task allocation which reduces maximum chip temperature can achieve large power saving. On the other hand, when temperature constraint is loose, the convective resistance does not have to be small to satisfy the constraint. In this case, the curve slope is flat and the difference in convective resistance does not affect the fan power and overall power consumption significantly. Therefore, different task allocations achieve similar overall system power consumptions.

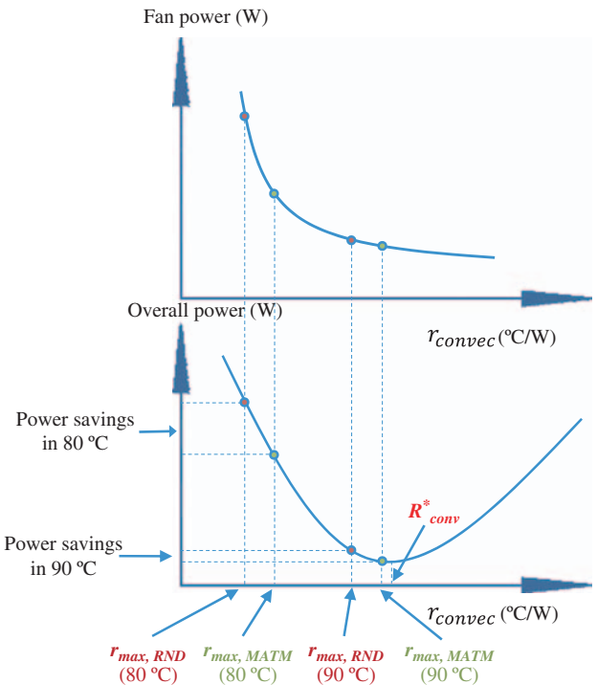

Fig. 11. Power consumption against convective thermal resistance curve.

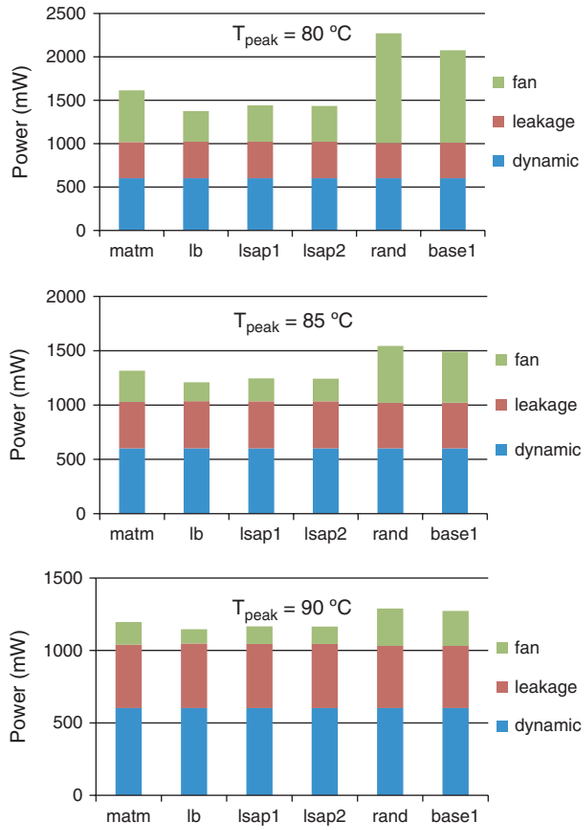

Fig. 12. The overall power consumption break down for different temperature constraints.

If we further relax the temperature constraint so that the r_{max} of both random and MATM allocations are located to the right side of R^*_{conv}, the MATM allocation will not give any power saving over the random allocation as both of them can work at the optimal tradeoff point.

Figure 12 shows each component in the overall system power consumption. The fan power consumption plays an important part when temperature constraint is strict. It accounts for 25.6% to 49.6% of total consumption for different allocation policies when the temperature constraint is 80 °C. And the lower the peak temperature a policy could reach, the lower the fan power it will consume. When the constraint is relaxed, the share of fan power decreases. We also notice that the dynamic power stays the same for all constraints while the leakage power increase as the constraint is relaxed. This is because allowing higher maximum chip temperature will also increase the average chip temperature, therefore the leakage power increases. We also notice that MATM based task allocation has higher leakage power consumption compare to the rand and base1 policy. This is because, in order to maintain the same maximum chip temperature, the higher fan speed needed for rand and base1 policy makes its average temperature lower and hence it consumes less leakage power. However, after combining the fan power, the MATM based allocation still has lower total power consumption.

6.3. The Impact of Initial Task Allocation and Multi-Hop MATM

In the previous experiments, we assume that the initial task mapping is randomly generated. Therefore, the high power tasks and low power tasks are evenly distributed across the system, i.e., in each area of the system, the number of hot tasks and cool tasks are roughly the same. In this experiment, we test an extreme case, which has high concentration of high power tasks in a small area in the initial mapping. The further a processor is away from this area, the higher probability that it will be assigned to a low power task. In our experiment, the "hot area" is located at the corner of the chip.

Table IV presents the convective resistances that are required to keep the peak chip temperature under 80 °C and the average chip temperature. Four algorithms are compared, two global policies, the MATM distributed policy and the best random mapping policy. Comparing the results in Section 6.1, we can see that the performance of the global policy is not affected by the initial task mapping. On the other hand, the performance of the distributed policy is significantly affected by the initial task allocation and it performs much worse under this extreme case. This is because the distributed policy relies on a rippling process to pass out high power tasks from the hot area. However, this rippling process is blocked by those cores located at the center of the chip. Therefore, most hot tasks cannot be delivered from the initially hot corner to other cores with strong heat dissipation ability located at the other corners and boundaries of the chip. In this case, the allocation obtained by the distributed policy is far from the optimal. Please note, in this corner case, because MATM needs the minimum r_{conv} to keep peak temperature under constraint, it actually results in the lowest average temperature.

Table IV. Comparison of convective resistance and average temperature when peak temperature constraint is 80 °C under extreme initial distribution.

Policy	LSAP-1 (R_{conv}(°C/W)/T_{avg}(K))	LSAP-2 (R_{conv}(°C/W)/T_{avg}(K))	MATM (R_{conv}(°C/W)/T_{avg}(K))	Random (R_{conv}(°C/W)/T_{avg}(K))
Uniform	0.0134/350.62	0.0135/350.73	0.0094/345.14	0.0096/346.60
Cool triangle	0.0157/349.97	0.0158/350.05	0.0116/344.84	0.0133/346.75
Hot triangle	0.0101/349.46	0.0103/349.83	0.0075/345.40	0.0080/347.06
Norm	0.0120/348.86	0.0120/348.84	0.0091/345.04	0.0098/346.54
Inv norm	0.0133/350.32	0.0137/350.43	0.0089/344.14	0.0099/346.50

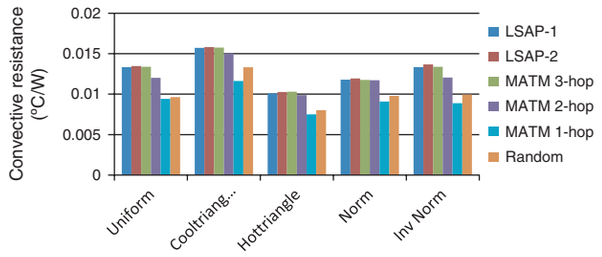

Fig. 13. Comparison of convective resistance for multi-hop MATM.

To find better task allocation in this extreme case, we slightly modify the distributed policy by allowing a processor to send migration requests to its far neighbors. Figure 13 show the required convective resistances if far neighbors can be included in the task migration. The MATM 1-hop refers to the original MATM policy where communication only happens between 1-hop neighbors, and the MATM 2-hop and 3-hop refer to the modified MATM policy where communication could happen between 2 and 3-hop neighbors respectively. As we can see, compare to the MATM 1-hop policy, in this extreme case, MATM 2-hop and 3-hop policy can save fan power by 55.1% and 62.2% respectively. And even in this extreme case, fan power savings achieved by the distributed MATM 3-hop policy is very close to the global policy.

Obviously, the reduced power does not come for free. Communication and exchanging tasks with 2 and 3-hop neighbors means consuming more energy on the on-chip networks. Here, we use migration distance to represent the energy overhead on the network. Because the energy consumed in the on-chip network is proportional to the communication distance.[11] For example, if a task is moved from core i to core j, then we define the migration distance to be the Manhattan distance between core i to core j in the number of hops. Figure 14 compares the migration distance for multi-hop MATM policy and the global policies. As expected, allowing exchanging tasks among far neighbors increase the migration distance significantly. On the other hand, compare to the global policies such as

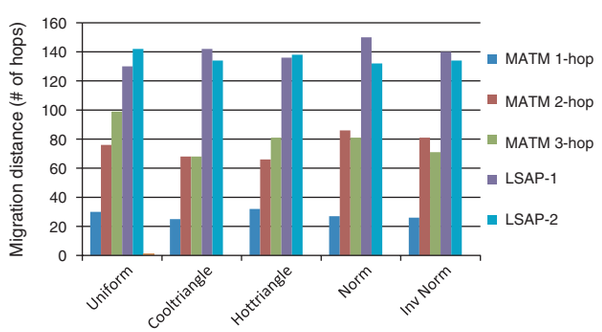

Fig. 14. Comparison of migration distance for multi-hop MATM policy and global policies.

LSAP-1 and LSAP-2, the 2-hop and 3-hop MATM policy still reduce migration overhead by 46% and 42%. Finally, we want to point out that in a real system, such extreme cases of initial task allocation can easily be avoided by simple random mapping of the tasks.

7. CONCLUSION

In this paper, we studied the impact of task mapping on the overall power consumption of a homogeneous many-core system. We formulated the peak temperature aware task mapping problem as a zero-one linear programming and proposed global heuristic algorithms as well as an agent based distributed task migration to solve this problem. Our agent based algorithm has good scalability as the number of processors increases. Experimental results show that our policy achieves large power savings compare to a random mapping policy and a baseline policy. It also produces solution close to the global policies while maintaining a low communication and control overhead.

Acknowledgment: This work was supported by the National Science Foundation under Grant CNS-1203986.

References

1. Tile Processor Architecture: Technology Brief. [Online]. Available: http://www.tilera.com/pdf/ProductBrief_TileArchitecture_Web_v4.pdf.
2. http://lpsolve.sourceforge.net/5.5/.
3. R. Ayoub, S. Sharifi, and T. Rosing, GentleCool: Cooling aware proactive workload scheduling in multi-machine systems, *Proc. Design Automation and Test in Europe*, March **(2010)**, pp. 295–298.
4. S. Borkar, Thousand Core chips—A technology perspective, *Proc. Design Automation Conference*, June **(2007)**, pp. 746–749.
5. D. Brooks, V. Tiwari, and M. Martonosi, Wattch: A framework for architectural level power analysis and optimizations. *Proc. Int. Symp. Computer Architecture*, June **(2000)**, pp. 83–94.
6. A.Coskun, T. Rosing, and K. Whisnant, Temperature aware task scheduling in MPSoCs, *Proc. Design Automation and Test in Europe*, April **(2007)**, pp. 1659–1664.
7. A. Coskun, T. Rosing, and K. Gross, Utilizing predictors for efficient thermal management in multiprocessor SoCs. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 28, 1503 **(2009)**.
8. J. Donald and M. Martonosi, Techniques for multicore thermal management: Classification and new exploration, *Proc. Int. Symp. Computer Architecture*, June **(2006)**, pp. 78–88.
9. T. Ebi, M. Al Faruque, and J. Henkel, TAPE: Thermal-aware agent-based power economy for multi/many-core architectures, *Proc. Int. Conf. on Comput.-Aided Design* (ICCAD), November **(2009)**, pp. 302–309.
10. J. Howard, S. Dighe, Y. Hoskote, S. Vangal, D. Finan, G. Ruhl, D. Jenkins, H. Wilson, N. Borkar, G. Schrom, F. Pailet, S. Jain, T. Jacob, S. Yada, S. Marella, P. Salihundam, V. Erraguntla, M. Konow, M. Riepen, G. Droege, J. Lindemann, M. Gries, T. Apel, K. Henriss, T. Lund-Larsen, S. Steibl, S. Borkar, V. De, R. Van Der Wijngaart, and T. Mattson, A 48-Core IA-32 message-passing processor with DVFS in 45 nm CMOS, *Proc. Int. Solid-State Circuits Conf.* (ISSCC), February **(2010)**, pp. 108–109.
11. J. Hu and R. Marculescu, Energy-aware mapping for tile-based NoC architectures under performance constraints, *Proc. Asia and South Pacific Design Autom. Conf. (ASP-DAC)*, January **(2003)**, pp. 233–239.

12. C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler, and T. Keller, Energy management for commercial servers. *IEEE Computer* 36, 39 **(2003)**.

13. Y. Liu, R. Dick, L. Shang, and H. Yang, Accurate temperature-dependent integrated circuit leakage power estimation is easy, *Proc. Design Automation and Test in Europe*, April **(2007)**, pp. 1526–1531.

14. J. Moorey, J. Chasey, P. Ranganathanz, and R. Sharma, Making Scheduling cool: Temperature-aware workload placement in data centers, *Proc. Annual Conference on USENIX Annual Technical Conference*, April **(2005)**, pp. 5–18.

15. F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, and D. Atienza, Thermal balancing policy for streaming computing on multiprocessor architectures, *Proc. Design Automation and Test in Europe*, March **(2008)**, pp. 734–739.

16. E. Pakbaznia, M. Ghasemazar, and M. Pedram, Temperature-aware dynamic resource provisioning in a power-optimized datacenter, *Proc. Design Automation and Test in Europe*, March **(2010)**, pp. 124–129.

17. D. Shin, N. Chang, J. Choi, S. Chung, and E. Chung, Energy-optimal dynamic thermal management for green computing, *Proc. Int. Conf. Computer-Aided Design*, November **(2009)**, pp. 652–657.

18. K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, Temperature-Aware Microarchitecture: Modeling and Implementation. *ACM Trans. Architecture and Code Optimization* 1, 94 **(2004)**.

19. Q. Tang, S. Gupta, and G. Varsamopoulos, Energy-Efficient, Thermal-Aware Task Scheduling for Homogeneous, High Performance Computing Data Centers: A Cyber-Physical Approach. *IEEE Trans. Parallel and Distributed Syst.* 19, 1458 **(2008)**.

20. S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Lyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, An 80-Tile 1.28 TFLOPS network-on-chip in 65 nm CMOS, *Proc. Int. Solid-State Circuits Conf.*, February **(2007)**, pp. 98–589.

21. Y. Wang, K. Ma, and X. Wang, Temperature-constrained power control for chip multiprocessors with online model estimation, *Proc. Int. Symp. Computer Architecture*, June **(2009)**, pp. 314–324.

22. Y. Ge and Q. Qiu, Task allocation for minimum system power in a homogenous multi-core processor. *Int. Green Computing Conf.*, August **(2010)**, pp. 299–306.

23. Y. Ge, P. Malani, and Q. Qiu, Distributed task migration for thermal management in many-core systems. *Proc. Design Automation Conference*, June **(2010)**.

24. S. Manoj, K. Wang, and H. Yu, Peak power reduction and workload balancing by space-time multiplexing based demand-supply matching for 3D thousand-core microprocessor, *Proc. Design Automation Conference (DAC)*, May **(2013)**.

25. Y. Cui, W. Zhang, and H. Yu, Distributed thermal-aware task scheduling for 3D network-on-chip, *2012 IEEE 30th International Conference on Proc. Computer Design (ICCD)*, September **(2012)**.

26. W. Huang, M. Allen-Ware, J. B., Carter, E. Elnozahy, H. Hamann, T. Keller, C. Lefurgy, J. Li, K. Rajamani, and J. Rubio, TAPO: Thermal-aware power optimization techniques for servers and data centers, *Proc. International Green Computing Conference and Workshops (IGCC)*, July **(2011)**, pp. 1–8.

27. R. Marculescu, U. Ogras, L. Peh, N. Jerger, and Y. Hoskote, Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 28, 3 **(2009)**.

28. B. Shi, A. Srivastava, and A. Bar-Cohen, Hybrid 3D-IC cooling system using micro-fluidic cooling and thermal TSVs, *Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, August **(2012)**, pp. 33–38.

Yang Ge

Yang Ge *received his B.S. degree in telecommunication engineering from Zhejiang University, China in 2007, M.S. degree from the department of Electrical and Computer Engineering of Binghamton University, USA in 2009, and Ph.D. degree in Department of Electrical Engineering and Computer Science in Syracuse University, USA in 2012. He is currently an ASIC design scientist in Broadcom Corporation. His research interests include power and thermal analysis and optimization for multi and many-core system.*

Yukan Zhang

Yukan Zhang *received her B.S. degree in electrical engineering from Nankai University, China in 2006, and M.S. degree from the department of Electrical and Computer Engineering of Binghamton University, USA in 2009. She is currently working on her Ph.D. degree in Department of Electrical Engineering and Computer Science in Syracuse University, USA. Her research interests include energy harvesting and management for embedded systems.*

Qinru Qiu

Qinru Qiu *received her M.S. and Ph.D. degrees from the department of Electrical Engineering at University of Southern California in 1998 and 2001 respectively. She received her B.S. degree from the department of Information Science and Electronic Engineering at Zhejiang University, China in 1994. Dr. Qiu is currently an associate professor at the Department of Electrical Engineering and Computer Science in Syracuse University. Before joining Syracuse University, she has been an assistant professor and then an associate professor at the Department of Electrical and Computer Engineering in State University of New York, Binghamton. Her research areas are energy efficient computing systems, energy harvesting real-time embedded systems, and neuromorphic computing. She has published more than 50 research papers in referred journals and conferences. Her works are supported by NSF, DoD and Air Force Research Laboratory.*