A Multi-Agent Framework for Thermal Aware Task Migration in Many-Core Systems

Yang Ge, *Student Member, IEEE*, Qinru Qiu, *Member, IEEE*, and Qing Wu, *Member, IEEE*

***Abstract*—In deep submicrometer era, thermal hot spots, and large temperature gradients significantly impact system reliability, performance, cost, and leakage power. As the system complexity increases, it is more and more difficult to perform thermal management in a centralized manner because of state explosion and the overhead of monitoring the entire chip. In this paper, we propose a framework for distributed thermal management in many-core systems where balanced thermal profile can be achieved by proactive task migration among neighboring cores. The framework has a low cost agent residing in each core that observes the local workload and temperature and communicates with its nearest neighbor for task migration and exchange. By choosing only those migration requests that will result in balanced workload without generating thermal emergency, the proposed framework maintains workload balance across the system and avoids unnecessary migration. Experimental results show that, our distributed management policy achieves almost the same performance as a global management policy when the tasks are initially randomly distributed. Compared with existing proactive task migration technique, our approach generates less hotspot, less migration overhead with negligible performance overhead.**

***Index Terms*—Distributed control, dynamic thermal management, multi-agent, prediction, task migration.**

I. Introduction

With the unprecedented number of transistors integrated on a single chip, the current multi-core technology may soon progress to hundreds or thousands of cores era [3]. Examples of such system are the 80-tile network-on-chip that has been fabricated and tested by Intel [28] and Tilera's 64 core TILE64 processor [1]. While the multicore or many-core technology delivers extraordinary performance, they have to face the significant power and thermal challenges.

The increasing chip complexity and power density elevate peak temperatures of chip and unbalance the thermal gradients. Raised peak temperatures reduce lifetime of the chip, deteriorate its performance, affect the reliability [27] and increase the cooling cost. Dynamic thermal management (DTM) approaches such as core throttling or stalling which are widely used in today's computer systems usually have negative impact on the performance. The adverse positive feedback between leakage power and raised temperature creates the potential of thermal runaway [27]. When mapped on a many-core system, diverse workload of applications may lead to power and temperature imbalance among different cores. Such temporal and spatial variations in temperature create local temperature maxima on the chip called the hotspot [11], [27]. An excessive spatial temperature variation, which is also referred to as the thermal gradients, increases clock skews and decreases performance and reliability.

Many dynamic thermal management techniques such as clock gating, dynamic voltage, and frequency scaling, thread migration have been proposed for multi-core systems. All these techniques aim to ensure the system running under a fixed safe temperature constraint [8], [9], [16], [19], [21], [22], [26], [29].

Most of these existing techniques are centralized approaches. They require a controller that monitors the temperature and workload distribution of each core on the entire chip and make global decisions of resource allocation. Such centralized approaches do not have good scalability. First of all, as the number of processing elements grows, the complexity of solving the resource management problem grows exponentially. Second, a centralized resource management unit that monitors the status and issues DTM commands to each core generates a huge communication overhead in many-core architecture, as communication between the central controller and cores will increase exponentially with the number of cores [12]. Such overhead will eventually affect the speed of data communication among user programs and also consume more power on the interconnect network. Finally, as the size and the complexity of the many-core system increase the communication latency between the central controller and the cores increases, this leads to a delayed response and sub-optimal control.

In this paper, we propose a framework of distributed thermal management where balanced thermal profile can be achieved by proactive thermal throttling as well as thermal-aware task migrations among neighboring cores. The framework has a low cost agent residing in each processing element (PE). The agent observes the workload and temperature of the PE while exchanging tasks with its nearest neighbors through negotiation and communication. The goal of the proposed task migration is to match the PE's heat removal capability to its workload (i.e., the average power consumption) and at the same time create a good mix of high power (i.e., "hot") tasks and low power (i.e., "cool") tasks running on it. As each agent monitors only the local PE and communicates with its nearest neighbors, the proposed framework achieves much better scalability than the centralized approach. We refer to the proposed technique as distributed thermal balancing migration (DTB-M) as it aims at balancing the workload and temperature of the processors simultaneously.

Manuscript received July 29, 2010; revised January 24, 2011 and May 16, 2011; accepted June 13, 2011. This work was supported by the National Science Foundation under Grant CNS-0845947.

Y. Ge and Q. Qiu are with the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY 13244 USA (e-mail: eroicaleo@gmail.com; qinru.qiu@gmail.com).

Q. Wu is with the Information Directorate of United States Air Force Research Laboratory, Rome, NY 13440 USA (e-mail: qwu2000@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TVLSI.2011.2162348

A steady-state temperature-based migration (SSTM) scheme as well as a temperature prediction-based migration (TPM) scheme are proposed in this paper. The first migration scheme considers the long term thermal behavior of tasks, and distributes tasks to PEs based on their different heat removal capabilities. The second migration scheme predicts the thermal impact of different workload combinations and adjusts the task allocation in a neighborhood so that all the PEs get a good mixture of hot tasks and cool tasks. The two migration schemes are complementary to each other with the first considers long term average thermal effect and the second considers short term temporal thermal variations. Both SSTM and TPM methods are proactive migration schemes. Together they provide progressive improvement that reduces thermal gradients and prevents thermal throttling events.

As part of the thermal management agent, a neural network-based temperature predictor is also proposed in this paper. It predicts the future peak temperature based on the workload statistics of the local PE and some preliminary information from the neighboring PEs. Comparing to the temperature predictors proposed in previous works [9], [31], our neural network predictor has several advantages. First of all, it only has to be trained once and after that the recall process has very low computation complexity. Second, because it takes the workload information as one of the input parameters, it can give accurate prediction right after task migration. This is the major difference between our prediction model and the previous prediction models [9] and [31] which need an online adaptation phase when workload changes. Finally, our model can be used to predict the temperature impact of a migration before the migration physically takes place, as long as the power consumption of the task to be migrated in or out is known. Therefore, the predictor is used not only to determine when to trigger a proactive task migration but also to evaluate whether a migration is beneficial.

The following summarizes the key contributions of the DTB-M thermal management framework.

1) No centralized controller is required in this framework. The distributed thermal management agent communicates and exchanges tasks only with its nearest neighbors. Therefore, the communication cost and migration overhead for each core does not increase as the number of PEs on the chip increases.
2) Comparing to the existing temperature prediction models [9], [31], the neural network-based peak temperature predictor works more robustly especially during the time when the workload changes, which usually happens after task migration.
3) Comparing to the existing proactive thermal-aware task migration, the proposed migration policy results in lower peak temperature and reduces the number of thermal throttling events. Experimental results show that, in average, the DTB-M reduces the occurrence of hotspots by 29.8% at 0.98% performance overhead compared to the proactive thermal balancing (PTB) algorithm proposed in [9]. Furthermore, the DTB-M also has much lower migration overhead due to its distributed nature.

Comparing to our original work in [34], this work provides the following two major extensions.

The first major extension of this paper is a thorough study of the performance of neural network model. The investigation covers three areas.

1) We varied the size (number of neurons) of the neural network model and compared their prediction accuracies. The results show that fairly good prediction accuracy could be achieved with very small size neural network.
2) We also examined the impact of input feature set selection on the prediction accuracy. The above analysis leads to an improved neural network model with better accuracy and computation complexity tradeoff than the one presented in our previous work [34].
3) We compared our neural network model with an improved auto-regression moving average (ARMA) prediction model proposed in [9]. The improvement is added in order to have a fair comparison as the original ARMA model does not consider as many input information as we do in the neural network model, and this impairs the accuracy. We test the accuracies of these two models not only on systems with stable workload but also on systems with dynamic workload where tasks start, complete and migrate from time to time.

The second major extension of the paper is the enriched experimental results section. We investigated the impact of different prediction models on the efficiency of the proposed migration policy. We also evaluated the performance of the SSTM and TPM policies separately in order to assess their individual contributions to the thermal management. The results show that the SSTM policy gives more hotspot reduction and leads to better system performance; therefore it should be assigned with higher priority during the runtime. However, using TPM following SSTM can give us extra reduction in hotspots and improvements in system performance. We further demonstrate the effectiveness of using distributed control by applying the same migration policy in a global manner. The results show that although in average the global policy has about 13% less hotspot than the distributed policy, its migration overhead is 58% higher. Finally, we compared our migration policy with the PTB policy proposed in [9].

The rest of this paper is organized as follows. Section II reviews the previous work. Section III gives the semantics of the underlying many-core system and the application model. We give an overview of our thermal management policy in Section IV, while the detailed prediction model and migration schemes are presented in Sections V and VI, respectively. Experimental results are reported in Section VII. Finally, we conclude this paper in Section VIII.

II. Related Work

Modern day microprocessers handle thermal emergencies through various DTM mechanisms. Techniques at microarchitecture level have been well explored in [11] and [27]. At system level, voltage/frequency scaling, task scheduling, task allocation, and thread migration can be combined to leverage the temperature reduction on MPSoCs. In [22], frequency assignment has been formulated as a convex optimization problem and optimum solutions can be solved offline. Online voltage/frequency scaling techniques often utilize a feedback

controller to adjust voltage/frequency settings. The authors of [32] use a linear quadratic regulator to adjust the frequency assignment online for thermal balancing. In [29], chip power consumption and operating temperature are controlled to a desired point by a multiple-input and multiple-output (MIMO) controller based on the model predictive control theory. Coskun *et al.* [8] proposed a light weight probability based scheduling method which could achieve better temporal and spatial thermal profile.

In a many-core system, the heat dissipation capability differs from processor to processor. In [33], an algorithm is proposed to map and schedule tasks based on the thermal conductivity of different processors. In [26], Sharifi *et al.* proposed a task allocation and frequency assignment algorithm which use exhaustive search to find a location and a voltage/frequency setting for incoming tasks to achieve energy saving and balanced temperature. Michaud *et al.* [19] proposed a clock gating and thread migration based method which maximizes system performance and minimize the number of migrations while maintaining the temperature under a desired constraint and guaranteeing fairness between threads. The throughput of an MPSoC system under a maximum temperature constraint has been studied in [24], and they derived an approximate analytic expression of system throughput depend on several parameters of interest.

Thermal management of on-chip interconnect network is addressed in [25]. Shang *et al.* first proposed an architecture thermal model for on-chip networks. Based on this model, they further proposed ThermalHerd, a framework which uses distributed thermal throttling and thermal aware routing to tackle thermal emergencies.

Proactive thermal management based on runtime task migration has been proposed in references [9] and [31]. Both of them predict the future temperature as a projection of the history temperature trace. Although these predictive models are very accurate in most circumstances, they have some limitations. First of all, both models have to be updated and adjusted at runtime. This could introduce adaption overhead. Second, both models predict the future temperature solely from the temperature history. For a system with frequent task migrations, history trace does not reflect future temperature because the workload changes dramatically. The predictor cannot give accurate prediction until it has adapted to the new workload which may take a long time.

Unlike the prediction model proposed in [9] and [31], our neural network-based prediction model can overcome the limitations mentioned previously. Our model does not rely on the history temperature. Instead it reveals the relation between temperature and workload. It is trained offline; and does not need an online adaption phase. As the model is trained separately for each core on the chip, it inherently takes into account the core location and heat dissipation ability.

III. System Infrastructure

A tile-based network-on-chip (NoC) architecture [10] is targeted here. Each tile is a processor with dedicated memory and an embedded router. It will also be referred to as core or PE 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*. The proposed DTB-M algorithm moves tasks among nearest neighbors in order to reduce overhead and minimize the impact on the communication bandwidth.

In an NoC, the latency and energy for transferring a data packet from one PE to another is proportional to the number of hops along the path [14], [18]. If we consider the congestions, this relation could be super linear due to the buffering overhead at each router. Limiting the communication to nearest neighbors cuts the communication cost (including both latency and energy) by reducing the communication distances and eliminating congestions.

We assume an existence of temperature sensor on each core. A temperature sensor can be a simple diode with reasonably fast and accurate response [11].

We 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. This is a trend pointed out by some literatures in OS research for many-core and NoC systems [20], [23]. Examples of such system are Intel's single-chip cloud computing (SCC) platform [13] and research accelerator for multiple processors (RAMP) [30].

The proposed DTB-M algorithm is implemented as part of the OS-based resource management program which performs thermal-aware task migration. We assume that each core is a preemptive time-sharing/multitasking system. We focus on batch processing mode, where pending processes/tasks are enqueued and scheduled by the agent. Each task occupies an equal slice of operating time. Between two execution slices is the scheduling interval in which the agent performs the proposed DTB-M algorithm and the OS switches from one task to another. The scheduling intervals of different cores do not have to be synchronized. Because the context switch overhead is very small compare to the execution interval (e.g., in Linux), and our algorithm has very low overhead, we assume that the duration of the scheduling interval is negligible comparing to that of the execution interval.

In this work, we do not consider cores that support for simultaneous multithreading (SMT) because it is anticipated [3] and [15] that future many-core platform is composed of large number of weaker and smaller cores with less transistors and power consumption, therefore, they are more likely to be single-threaded cores. However, with some modification in the temperature prediction models, the same DTB-M algorithm could be applied to systems with SMT cores.

IV. Distributed Thermal Balancing Policy

In this section, we present the details of the DTB-M policy. Table I summarizes the notations that will be used in this paper.

As we mentioned before, each PE_i is a preemptive system and has a set of tasks LT_i. Each task occupies an equal slice of execution time t_{slice}. Between two execution intervals is the scheduling interval. Our DTB-M policy is performed in scheduling interval. The PE also switches from one task to the next task at the scheduling interval. It is assumed that each task in LT_i will be running for a relatively long period of time and its power consumption has been profiled or can be estimated. For example, it is reported in [5] that more than 95% accuracy can be achieved in power estimation using information provided by

TABLE I
LIST OF SYMBOLS AND THEIR DEFINITIONS

Symbol	Definition
LT_i	The list of tasks running on core i
$\|LT_i\|$	The number of tasks running on core i
$\tau_{i,}$	A task in LT_i
$P\tau_i$	The power of $\tau_{i,}$
T_i	Current temperature of core i
N_i	The set of nearest neighbors of core i
T_m	Temperature threshold to trigger the DTB-M algorithm
T_{diff}	Threshold to trigger thermal balancing
nt_{diff}	Threshold to trigger workload balancing
t_{slice}	Execution interval

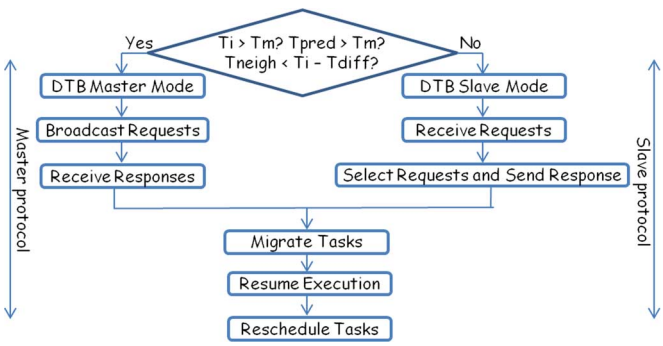


Fig. 1. Master-slave execution protocol.

performance counters that are available in many modern processors. In the rest of this paper, we refer to the power consumptions of all tasks in LT_i as the "*workload*" of PE_i and we refer to the different combinations of tasks in the LT_i as different "*workload patterns*" of PE_i. Both information can easily be observed by OS.

The DTB-M policy basically can be divided into 3 phases: temperature checking and prediction, information exchange and task migration. Fig. 1 shows the flowchart of the DTB-M execution in the ith core. A DTB-M agent could be in either master mode or slave mode. A DTB-M master initiates a task migration request while the DTB-M slave responds to a task migration request. A DTB-M agent is in slave mode by default. It will enter the master mode if and only if any of the following three scenarios are true.

1) The local temperature T_i reaches a threshold T_m in the last execution interval. In this case, hotspots are generated, and the DTB-M agent will first throttle the processor to let it cool down before continue to execute.
2) The predicted future peak temperature exceeds the threshold T_m and the current peak temperature is larger than $T_m - \delta$, where δ is a temperature margin. Note that we do not take actions unless the difference between the current peak temperature and the threshold is less than the margin.
3) The temperature difference between the local core and the neighbor core exceeds the thermal balancing threshold T_{diff}.

Any of the above three scenarios could cause adverse effects. The first two scenarios indicate (potential) hotspots generation while the last scenario indicates high thermal gradients. Therefore, a task migration request will be initiated.

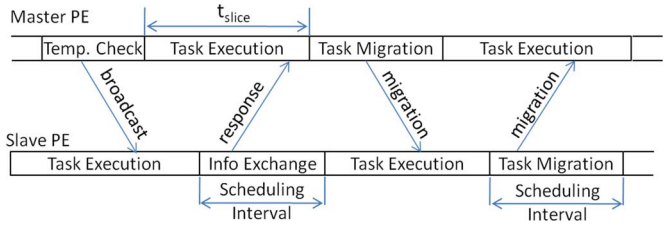


Fig. 2. Master-slave communication.

A DTB-M master sends task migration requests to its nearest neighbors. Because the scheduling intervals in all processors are not synchronized, the requests are not likely to be checked and responded by the slave agents right away. On the other hand, because all cores adopt the same execution and scheduling interval, it is guaranteed that all slave agents will respond within one t_{slice} after the requests are issued.

The asynchronous communication between master and slave agents is explained by the example shown in Fig. 2. It shows a complete execution cycle of DTB-M policy starting from condition check phase to task migration. When an agent first enters its scheduling interval and becomes a master, it broadcasts a migration request in its neighborhood and then continues task execution.

The slave will not respond until it reaches the next scheduling interval, when it checks its message queue for incoming requests. If there is no request, the PE resumes normal execution in next time slice. In case of multiple master requests, the slave selects a master which has the highest average power consumption. Response to this master PE includes its ID, details of slave's workload, its recent operating temperature etc. The slave is then locked to this master until it is released by the master.

After receiving the response, the master decides which tasks to migrate during its next scheduling interval and sends the migration command to slave. The tasks are migrated from master to slave at this time. After sending a response, the slave ignores any possible incoming requests from other master agents until it receives the migration command from the original master. Tasks can be migrated from slave to master at this time, which marks the end of DTB-M policy cycle.

To make migration decisions, a master DTB agent considers both load balancing as well as thermal balancing. First, a load balancing process is triggered which migrates tasks one way to balance the workload between the master and the slave if the workload difference between them exceeds the threshold nt_{diff}, which is measured by $||LT_i| - |LT_j|| > nt_{\text{diff}}, j \ni N_i$. The detailed workload balancing policy is presented in Section VI-D. If there is no workload imbalance, then the thermal balancing process is triggered.

The main idea of the DTB-M policy is to exchange tasks between neighboring PEs, so that each PE can get a set of tasks that produces fewer hotspots. The DTB-M policy is composed of two techniques. Both of the techniques have quadratic complexities to the number of tasks in the local task queue. The first technique is a SSTM. It distributes tasks to cores based on their different heat dissipation abilities. The second technique is a TPM, which relies on predicted peak temperatures of different task combinations to make migration decisions. It ensures that each core can get a good mixture of high power and low power tasks without having thermal emergency. The two techniques

are complementary to each other with the SSTM focuses on long term average thermal effect and the TPM focuses on short term temporal variations. The main computation of the SSTM is performed by the masters while the main computation of the TPM is performed by the slaves.

The DTB-M agent is not a separate task but resides in the kernel code. For example, it can be integrated with the Linux task scheduler, which will be called each time when a task finishes its current time slice and gives up the CPU.

V. Temperature Prediction Model

Instead of projecting the future temperature based on a sequence of history temperatures, we model the peak temperature of a processor as a function of a set of features collected from local processor and its neighboring processors within a history window, and approximate this function using a neural network. Our feature set includes not only the temperature information but also the workload information. Because the relation between temperature and workload is relatively stable when the layout and packaging style of the chip is given, the neural network needs to be trained only once.

The rest of the section is organized as follows. Section V-A presents our neural network prediction model. Section V-B extends the ARMA prediction model proposed in [9] and Section V-C compares the performance of the two prediction models.

A. Neural Network-Based Temperature Prediction Model

The peak temperature predictor will be used in the temperature checking/prediction phase to determine if a master mode DTB-M will be triggered and also in the information exchange phase to find out if a TPM migration is beneficial or not. Therefore, it should not only give accurate peak temperature estimation when the PE continues the current workload pattern, but also project the temperature change before dramatic workload changes.

Temperature prediction in a timesharing/multitasking system is challenging. For example, Linux system makes context switch every tens of milliseconds. Different tasks have different power consumptions and therefore display different thermal characteristics. When running the combination of these tasks, the temperature of a PE would oscillate rapidly, making accurate temperature prediction difficult. Fortunately, we observed that the local peak temperature for a given set of tasks changes much slower compared to the instantaneous temperature. For example, Fig. 3 shows a 12 s long temperature trace of a processor time-multiplexed by a set of tasks randomly picked from SPEC 2000 benchmarks. We sampled the trace at a time step of 50 ms. We can see that, for a given workload pattern (i.e., a given combination of tasks in the ready queue), the instantaneous temperature variation of the PE can be as large as 8 °C and it changes rapidly while the peak temperature changes much slower and the variation is less than 2 °C. Similar observation has been reported in [6]. Our second observation is that the peak temperature strongly depends on the task combinations running on the PE. As shown in Fig. 3, there are five different workload patterns running on the PE. The temperature curve exhibits different characteristics during each workload pattern and the local peak temperature is

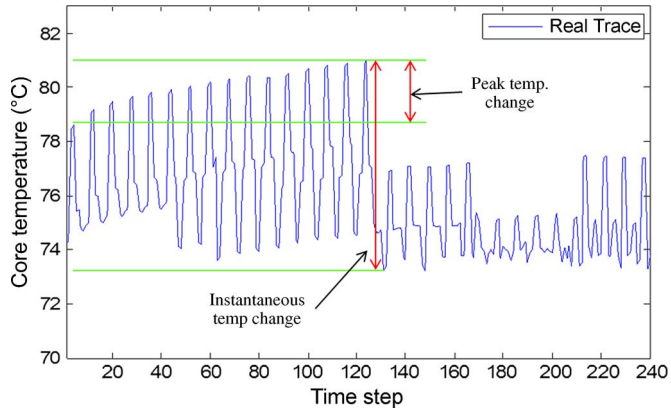


Fig. 3. Example of instantaneous and peak temperature change.

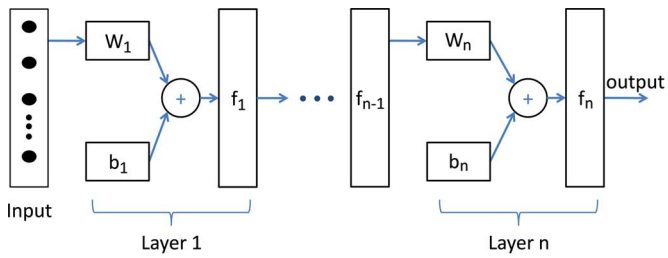


Fig. 4. Neural network structure.

changing considerably from one pattern to another. Because a high peak temperature causes the thermal emergency, here we are interested in predicting the PE's peak temperature in the near future given the set of tasks (i.e., the workload pattern) on this processor.

We adopt the neural network model for the peak temperature prediction. Neural network has been widely used in pattern recognition and data classification because of their remarkable ability to extract patterns and detect trends through complex or imprecise data [2]. It is composed of a number of interconnected processing elements (i.e., neurons) working together to solve a specific problem. A neural network model can be trained through a standard learning process. After the training process, the model can be used to provide projections on the new data of interest.

The general architecture of a neural network model is shown in Fig. 4. The model may have several layers, and each layer implements the function $\boldsymbol{a} = f(\boldsymbol{W}\boldsymbol{I}+\boldsymbol{b})$, where $f(\cdot)$ is a transfer function, $\boldsymbol{W}$ is a weight matrix, $\boldsymbol{b}$ is a bias vector, and $\boldsymbol{I}$ and $\boldsymbol{a}$ are input and output vectors. The sizes of $\boldsymbol{W}$ and $\boldsymbol{b}$ are m-by-s and m-by-1, where s is the dimension of the input vector and m is the number of neurons in this layer. Consequently, the output vector $\boldsymbol{a}$ has the dimension m-by-1. For a multi-layer neural network, the relation between the input of the model and the output of the model can be characterized by (1), where f_k is the transfer function, $\boldsymbol{W}_k$ is the weight matrix, and $\boldsymbol{b}_k$ is the bias vector for the kth layer, respectively, and $\boldsymbol{p}$ is the input vector to the neural network

$$a = f_n\left(\boldsymbol{W}_n f_{n-1}\left(\cdots f_1(\boldsymbol{W}_1\boldsymbol{p}+\boldsymbol{b}_1)\right)+\boldsymbol{b}_n\right). \qquad (1)$$

The training of the neural network predictor is an offline procedure and needs to be done only once. Therefore, here we only consider the complexity of the recall procedure, which is used online to predict the peak temperature. The recall procedure has

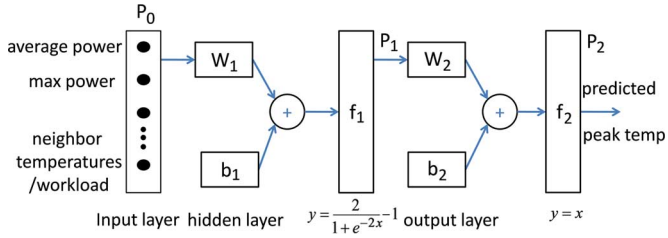


Fig. 5. Neural network predictor architecture.

very low complexity, which involves only $ms + m$ multiplications and $ms + 2m + 1$ additions.

In this paper, a two-layer neural network as shown in Fig. 5 is applied for peak temperature prediction. It has a hidden layer, and an output layer. There is only one neuron in the output layer because the output has to be a scalar variable. The number of neurons in the hidden layer should be selected to provide a good balance between the prediction accuracy and computing complexity. Later in this section we will show that, one neuron in the hidden layer is enough to provide good prediction accuracy. We use *tansig* and *purelin* functions as the transfer functions for the hidden layer (f_1) and the output layer (f_2), respectively. They are defined as the following two equations:

$$tansig(x) = \frac{2}{1 + \exp(-2x)} - 1 \tag{2}$$

$$purelin(x) = x. \tag{3}$$

A set of features relevant to the peak temperature prediction are selected as the inputs to the neural network. They can be divided into two categories, i.e., features collected from local processor and features collected from neighbor processors. The local feature consists of two variables. They give the average power consumption and maximum power consumption of tasks running on the local processor. For the ith core, they can be calculated as $\sum_{\tau_i \in LT_i} P_{\tau_i}/|LT_i|$, and $\max_{\tau_i \in LT_i}(P_{\tau_i})$, respectively. The feature set for neighbor information consists of three variables for each neighboring processor. They specify the recent highest temperatures in a history window, the average power consumption and the maximum power consumption of each neighboring processor. Overall there will be $3n + 2$ input variables to the neural network where n is the number of neighboring processors of the current PE.

A neural network-based peak temperature predictor is trained for each processor. The training process uses the fast and memory efficient Levenberg-Marquardt algorithm [16] provided by MATLAB neural network toolbox. The training set is generated by running 600 groups of randomly picked synthetic workload on our many-core simulator and recording the peak temperature of each PE for different workloads. Each group of workload consists of 144 artificially generated software programs randomly distributed across the many-core system. Each software program in the training workload has constant power consumption. Note that these artificially generated software programs are used only for the training purpose. All our experiments in the rest of the paper are based on benchmarks randomly picked from SPEC 2000, Mediabench, and MiBench. There is no overlapping between our testing set and training set. More details on the testing programs are provided in Section VII-A. Because the neural network model is trained for each core on the chip separately, these models are able to capture the core to core process variations.

TABLE II
PREDICTION ACCURACY VERSUS THE SIZE OF NEURAL NETWORK

Order	1	2	3	4	5	7	10	15
MSE	0.068	0.225	0.040	0.038	0.090	0.044	0.045	0.048
Avg. err.	-2e-05	-0.0020	-0.0017	-0.0013	0.0077	-0.0030	-0.0018	0.0010

It is important to point out that the neural network model is based on an assumption that the peak temperature of a core is a deterministic function of all the features aforementioned plus some white noise. A training set that covers all possible feature settings will yield the best model. Therefore, the longer training set gives better training quality. However, it also increases the training time. The size of our training set (i.e., 600 vectors) is selected for a balanced training time and quality.

In general, the accuracy of the prediction can be improved by adding more neurons in the hidden layer. However, this will also increase the complexity of training and recall. Experiments have been conducted to evaluate the sensitivity of the prediction accuracy to the size of the neural network. Table II gives the relation between the size of the neural network and its accuracy for the peak temperature prediction. The first row specifies the number of neurons in the hidden layer while the second row gives the *Mean Square Error (MSE)* of the estimation. When there is 1 neuron in the hidden layer, the MSE is 0.068. Further increasing the value of m will not improve the accuracy significantly but introduce higher computation complexity. Therefore, we set m equal to 1 for all PEs.

Because the complexity of the neural network is proportional to the size of its input vector, next, we investigate the effect of feature selections on the prediction accuracy in order to find out the set of features that gives the best tradeoff between prediction accuracy and computing complexity. We divide the input into the following four groups: 1) local average power; 2) local maximum power; 3) neighbor temperature information; and 4) neighbor power information. We carried out extensive random simulations of a 6-by-6 many-core processor to find out how the feature selections can affect the peak temperature prediction. Details of the simulator are provided in Section VII.

Fig. 6 gives the MSE and the input size of neural network models based on different combinations of feature groups. It is not surprising that including all four feature groups can result in the most accurate model and the smallest MSE, which is 0.041. If only self power and neighbor temperature (i.e., feature groups 1, 2, and 3) are considered, the MSE is 0.433. Finally, if the model only takes neighbor information (i.e., feature groups 3 or 4) as the input, the derived model is most inaccurate and the MSE can be as high as 4.977. Therefore, in this work, we build our neural network based on the entire four feature groups.

Unlike other prediction models [9] and [31], we do not invoke the prediction at every time step. Instead, the predictor will be invoked when the core temperature exceeds the predicted value or when the workload pattern in the PE changes. Note that the workload pattern is determined by the set of tasks in the current ready queue. It will be changed if a task is generated, completed or migrated. These events can be monitored by the OS. For a task with several phases that have different power and thermal characteristics, we consider each phase a single task.

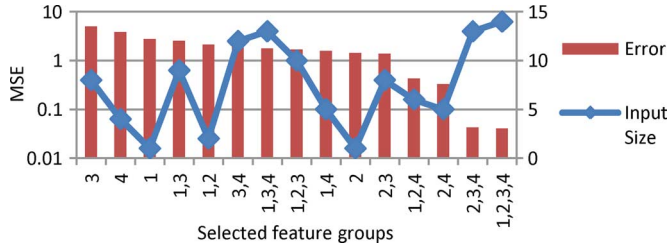


Fig. 6. Prediction error for neural networks based on different feature groups.

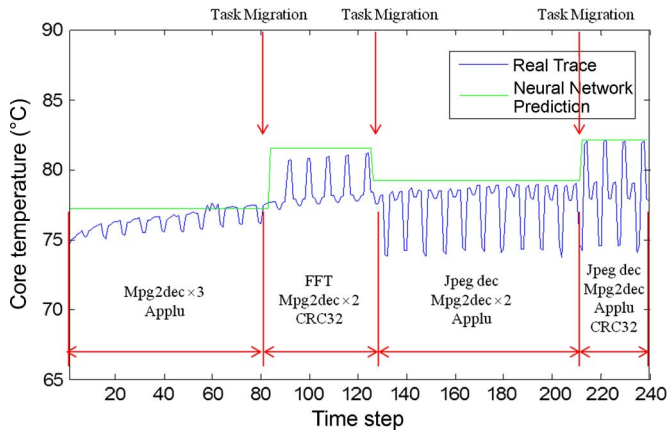


Fig. 7. Temperature prediction of neural network model.

New predictions will be made whenever a phase change is detected. Techniques for program phase change detection can be found in [7].

As an example, Fig. 7 shows the temperature trace of a PE and the predicted peak temperature given by the neural network. The temperature trace is generated by running several CPU benchmarks on our many-core simulator. During 12 s simulation time, tasks migrate, start, or complete randomly. The workload information at different time is denoted at the bottom of the figure. While the blue line gives the trace of the real temperature, the green line gives the predicted peak temperature. As we can see, the predictor is invoked every time the workload pattern changes and it is able to track the peak temperature accurately.

B. Generalized ARMA Prediction Model

In [9], Coskun *et al.* proposed to utilize the auto-regression moving average model to predict a PE's future temperature based on the previous temperature trace. The model is given by (4), where y_t is the temperature at time t, e_t is the prediction error, a_i and c_i are the coefficients. It consists of an auto-regressive (AR) part up to order p, which is on the left side of the equation, and a moving average (MA) part up to order q, which is on the right side of the equation. To utilize the ARMA model, we need to first identify the order of the model, and then compute the coefficients using least square fitting, and finally check the residuals to ensure the validity of the parameters

$$y_t + \sum_{i=1}^{p}(a_i y_{t-i}) = e_t + \sum_{i=1}^{q}(c_i e_{t-i}). \quad (4)$$

This model works very well when the temperature changes smoothly or there is a repeated pattern in the temperature change. However, it has two major limitations. First, in a multitasking system where threads start, finish, and migrate

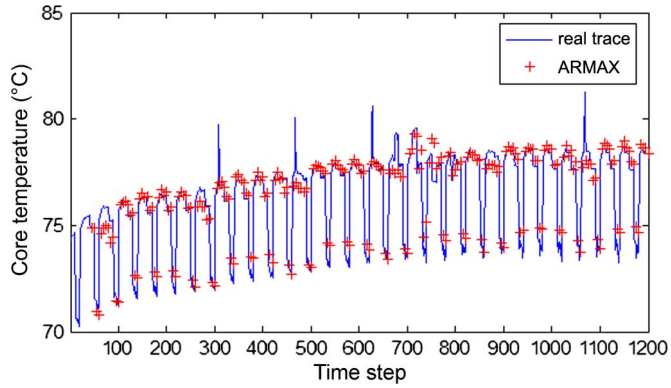


Fig. 8. Temperature prediction of ARMAX model.

dynamically, the adaptation time for the ARMA model is overwhelming. For example, in our experiment, we observed that the adaptation can be more than 50% of the execution time. Second, as we mentioned earlier, we are not only interested in predicting the future temperature when the current workload pattern continues, but also like to predict the temperature for a new workload pattern that has not physically been executed in order to assess the potential benefits of a task migration. This is not achievable using the ARMA model. While the first limitation is a fundamental issue related to all auto regression-based predictors, the second limitation can be improved by including some workload information in the original ARMA model.

In order to obtain a fair comparison between our neural network model and the existing prediction model, we extend the ARMA model to include the workload information as the exogenous inputs. The new model will be referred to as ARMAX [17] (i.e., auto-regressive moving average with exogenous inputs). It is described by (5), where u_t is the average power consumption of the task running at time t and b_i is the coefficient. Because we have already included the history temperature in the model, the input part could be reduced to only one item, i.e., $b_i u_{i-1}$. Therefore, the next temperature of the PE depends on p history temperature samples and the power consumption of the task it is currently running

$$y_t + \sum_{i=1}^{p}(a_i y_{t-i}) = e_t + \sum_{i=1}^{q}(c_i e_{t-i}) + \sum_{i=1}^{r}(b_i u_{t-i}). \quad (5)$$

Fig. 8 shows the temperature trace of a PE in a 6 × 6 many-core system obtained from Hotspot simulation and the temperature prediction made by the ARMAX model. The PE is time multiplexed by four tasks. Their execution order is fixed; therefore, the temperature trace shows a rough periodic pattern. Similar to [9] we set p and q to 8 and 0, respectively. The trace is 12 s long; we sampled 2 data points for each time slice and collected 240 temperature sample data. The results show that the final prediction error (FPE) [9] is 0.0265.

C. Comparing the Neural Network Predictor With ARMA Predictor

We compare our neural network based temperature predictor with ARMAX-based predictor. Fig. 9 shows a sequence of simulated temperature trace and the predicted temperature from the neural network and ARMAX models. The PE is initially running 4 tasks, after 3.25 s the PE exchanges its high power task with a

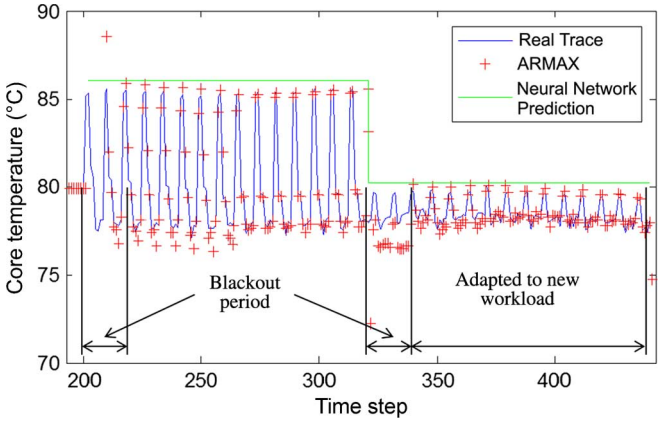


Fig. 9. Adaption ability of ARMAX model and neural network model.

TABLE III
AVERAGE POWER AND STEADY-STATE TEMPERATURE OF CPU BENCHMARKS

No.	1	2	3	4	5	6	7	8	9
Bench marks	crc32	mp2 enc	mp2 dec	fft	applu	mesa	bzip2	jpeg dec	jpeg enc
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

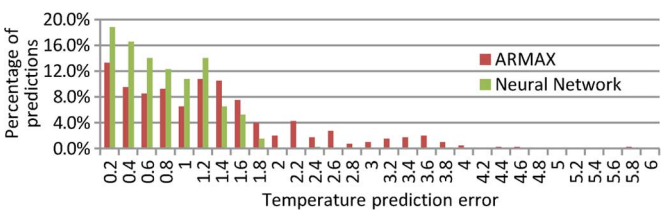


Fig. 10. Comparison of peak temperature prediction error.

low power task running on its nearest neighbor. As we can see, the neural network predictor adjusts its prediction to the correct level immediately after the migration while the ARMAX model takes more than 0.2 s to adapt to the right value. We refer this adaptation time as black-out period as the prediction results are not usable during this period of time.

We further compare the two models' capabilities to estimate the potential thermal impact before the migration. We simulate a 36-core system with 144 tasks randomly selected from real benchmarks listed in Table III. Approximately 200 migrations are randomly generated. Fig. 10 shows the absolute prediction error of the neural network model and the ARMAX model.

As shown in the figure, the average prediction error of the neural network model is 0.67 °C and the maximum prediction error is 2.5 °C. The average prediction error of the ARMAX is 1.2 °C which is 79% higher than that of the neural network predictor while its maximum error is 5.8 °C, which is 132% higher than that of the neural network model. For 99.75% of time the prediction error of the neural network is under 2 °C, while for 20% of time the prediction error of the ARMAX is above 2 °C. The difference is mainly because the ARMAX model has to take some time to adapt to the new workload after migration, and cannot make accurate prediction immediately.

Please note that the testing programs used in our experiments are different from the training programs. Our training set is artificially-generated programs with constant power consumptions. And the testing set consists of real benchmarks. However, the training set and testing set do share some similarity in those general features used for temperature prediction. For example, the ranges of power consumptions of the applications in the training and testing sets are very close. As long as two workloads have the same feature, their peak temperatures will be close to each other. Because the selected feature set is not extremely large, the 600 training vectors give reasonable coverage of possible scenarios. However, a larger training set can lead to more accurate model.

VI. DISTRIBUTED TASK MIGRATION POLICY

In this section, we present our distributed task migration policy. Section VI-A discusses the SSTM policy. Section VI-B discusses the TPM policy. Both of the SSTM and TPM have $O(n^2)$ time complexities, where n is the number of tasks in local ready queue of a processor. Section VI-C shows how to combine the two migration algorithms together. Finally, Section VI-D presents the workload balancing algorithm.

A. SSTM

Due to variant heat dissipation abilities, a task running on different processors have different steady state temperatures. The SSTM policy balances high power tasks and low power tasks among neighbor PEs to lower the average steady state temperature of the whole chip. It considers the lateral heat transfer between neighbor PEs and their different heat dissipation capabilities.

Before introducing the SSTM policy, we first give some definitions. We use n to denote the number of all thermal nodes in the system, including those in the heat sink layer and heat spread layer, and N to denote the number of processors in the system. The relation between n and N is determined by the equation $n = 2 \times N + 14$ [24]. We use TSS_i and P_i to denote the steady-state temperature and average power consumption of node i. P_i is 0 if node i belongs to the heat sink layer or heat spread layer. The vectors of TSS_i and P_i, where $1 \leq i \leq n$, are denoted as $\boldsymbol{TSS}$ and $\boldsymbol{P}$. When the system reaches the steady state, for each thermal node, its temperature is a linear function of power consumptions $P_1, P_2, \ldots, P_n$. The relation can be represented by the following equation:

$$\boldsymbol{TSS} = \boldsymbol{G}^{-1}\boldsymbol{P} \tag{6}$$

where $\boldsymbol{G}^{-1} = [g_{ij}]$ is the inverse of thermal conductance matrix $\boldsymbol{G}$. We simplify (6) by keeping only the thermal nodes related to the PEs

$$\begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} \tag{7}$$

where N is the number of processors, and $D_i = \sum_{j=N+1}^{n} g_{ij} \cdot P_j$ is a set of constants, because the power (P_j) of those nodes not related with processors do not change. The coefficients g_{ij} and D_i $1 \leq i, j \leq N$ can be obtained by offline analysis. Equation (7) shows that the steady state temperature of each PE is a linear function of average power consumptions on other PEs and increasing or reducing the power consumption of one

PE will have an impact on the steady state temperature of all other PEs.

Assume that PE_i and PE_j exchange some tasks, and their average power consumptions altered by ΔP_i and ΔP_j, respectively. Using (7), the total steady-state temperature change of all processors after task migration can be calculated as

$$\sum_{k=1}^{N} \Delta T_k = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j \tag{8}$$

where G_i and G_j are the sums of the ith and jth column of the thermal conductance matrix, i.e., $G_i = \sum_{m=1}^{N} g_{mi}$, $G_j = \sum_{n=1}^{N} g_{nj}$. Because the thermal conductance matrix of a chip does not change once the hardware is given, the values of G_i and G_j are constants and can be pre-characterized. Overall, it takes only two multiplications and one addition to calculate $\sum_{k=1}^{N} \Delta T_k$. As we mentioned earlier, the goal of the SSTM policy is to reduce the average steady state temperature of the many-core system. Therefore it exchanges task pairs to keep $\sum_{k=1}^{N} T_k$ decreasing, i.e., $\sum_{k=1}^{N} \Delta T_k < 0$.

The main computation of SSTM is done on the master PE. Algorithm 1 gives the SSTM policy. A master DTB-M agent in PE_i first forms all task pairs $(\tau_i, \tau_j), \tau_i \in LT_i, \tau_j \in LT_j, j \in N_i$ with $P_{\tau_i} > P_{\tau_j}$. Then for each task pair, (8) is evaluated. The task pair which gives the minimum ΔT_k is selected and tasks are swapped. The process continues until $\sum_{k=1}^{N} \Delta T_k > 0$ for all task pairs. In this way, the master can maintain fairness of workload and reduce its own operating temperature as well as the system's average steady state temperature.

Algorithm 1 SSTM

1. **for** each $\tau_i \in LT_i$
2. **for** each $\tau_j \in LT_j$, *s.t.* $j \in N_i, P_{\tau_i} > P_{\tau_j}$
3. $\Delta T_{ij} = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j$
4. **do** { $\Delta T_{min} = \min(\Delta T_{ij})$
5. **if** $(\Delta T_{min} < 0)$ $\quad \text{swap}(\tau_i, \tau_j)$
6. } **while** $(\Delta T_{min} < 0)$

B. Temperature Prediction Based Migration

The SSTM reduces the average steady state temperature of the whole chip. Although very effective, it has several limitations. First, it is possible that the SSTM moves all high power tasks in a neighborhood to one core whose G value is the minimum. Furthermore, if the G value of a core is less than the G value of all its neighbors, then using SSTM policy the core will not be able to exchange its high power task with a low power task in its neighborhood when it is overheated because this will increase the average steady state temperature of the chip.

Algorithm 2 TPM (Slave Process)

1. **Input**: LT_i (list of tasks on local PE) and LT_j (list of tasks on master PE)
2. Sort LT_i based on the ascending order of task power consumption
3. Sort LT_j based on the descending order of task power consumption
4. For each task $\tau_i \in LT_j$
5. For each task $\tau_j \in LT_j$
6. If $(P_{\tau_i} < P_{\tau_j})$
7. T_p = Predicted local peak temperature after task exchange;
8. If $(T_p < T_m)$ return to the master and exit;
9. Return NULL to the master and exit;

To escape from the above mentioned situation, we further propose the TPM. The TPM policy guides high temperature core to exchange tasks with its cooler neighbors as long as those task exchanges will not cause any thermal emergency in both cores. This is achieved by using the prediction model introduced in Section V.

Algorithm 2 shows the main computation of the TPM policy which is performed by the slave DTB-M agent. The algorithm scans the list of local tasks (i.e., LT_i) based on the ascending order of task power consumption and the list of tasks on the master PE (i.e., LT_j) based on the descending order of task power consumption. For each task pair $\tau_i \in LT_i$ and $\tau_j \in LT_j$, if the power consumption of the local task is lower than that of the remote task, the slave DTB-M agent employs the neural network based predictor to determine whether the local peak temperature will exceed the thermal threshold T_m after τ_i and τ_j are exchanged. The algorithm stops when first such task pair is found. The task pair is returned to the master DTB-M agent as an offer for potential task migration. Because of the way that the $LT_i$and LT_j are sorted, this offer specifies the highest power task that can be taken from the master PE and the lowest power task that will be given to the master PE without generating any thermal problem.

On the master side, algorithm 2.1 is executed. Upon receiving all offers from its neighbors, the master agent selects the offer that enables it to move out the task with the highest power consumption. If there is a tie, then it further selects the offer that enables it to move in the task with the lowest power.

Algorithm 2 TPM (Master Process)

1. Input: $S = \{(\tau_i, \tau_j) | (\tau_i, \tau_j) \text{are offers from neighbors}\}$
2. Select the offer $(\tau_i, \tau_j) \in S$ whose P_{τ_i} is the maximum
3. If there is a tie, select the offer $(\tau_i, \tau_j) \in S$ whose P_{τ_i} is the minimum

C. Combined Migration Policy

As discussed in the previous sections, the SSTM algorithm reduces the overall chip temperature by considering the thermal conductance of the chip. So that in a neighborhood, high power tasks can quickly be moved to the PEs that have better heat dissipation abilities, while low power tasks can be moved to the PEs that are more easily to heat up. On the other hand, the TPM algorithm prevents a core with stronger heat dissipation in a neighborhood from being overheated by proactively exchanging its high power tasks with low power tasks in the neighborhood.

The proposed DTB-M policy is a combination of both SSTM and TPM. After the master DTB-M agent triggers a migration request, it waits for the response from the slaves. In this request, the master sends out the list of its local tasks. Once the slave receives the request, it performs the TPM algorithm (slave process). In the reply message, it sends TPM offer together with

the list of local tasks to the master. The master then performs SSTM to search for task pairs that, once exchanged, could bring down the average chip temperature. If such task pair is found, then the master will issue a task migration command. Otherwise it performs the TPM algorithm (master process).

We employ a simple technique to schedule the execution of tasks. All tasks in a PE's ready queue are sorted based on their average power consumption. The thermal aware scheduler will execute hot and cool tasks alternatively starting from the coolest and the hottest tasks, then the second coolest tasks and the second hottest, until all tasks have been executed once. It will then start a new round of execution again. This simple yet effective scheduling technique reduces the core temperature by interleaving hot and cool tasks.

D. Workload Balancing Policy

Workload balancing is triggered when a master PE_i finds the workload difference between itself and a slave PE_j exceeds the threshold nt_{diff}, that is $||LT_i| - |LT_j|| > nt_{\text{diff}}, j \ni N_i$. The goal of workload balancing is to maintain approximately equal number of tasks on each core and therefore improve worst case latency and response time.

The master will pick the slave which gives the maximum workload difference. Then, tasks are migrated one by one from the PE with more tasks to the PE with fewer tasks until their difference is less than or equal to one. In every migration, (8) is evaluated and the task which minimizes the $\sum_{k=1}^{N} \Delta T_k$ will be selected. It can be proved that if $G_i > G_j$ and $|LT_i| > |LT_j|$, the migration from PE_i to PE_j will start from the task with the highest power. On the other hand, if $G_i > G_j$ and $|LT_i| < |LT_j|$, the migration from PE_j to PE_i will start from the task with the lowest power.

VII. Experimental Results

We implemented a power trace driven behavioral simulator of a many-core system using C++. Hotspot [27] is integrated to our simulator to simulate the system thermal behavior. Although 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. We set the thermal sampling interval of Hotspot to 30 μs, in order to speed up the simulation without significantly reducing the accuracy.

We evaluated the proposed thermal management policy using both static workload and dynamic workload. The system performance is characterized by the number of completed jobs within a given period of time. We assume that the temperature threshold T_m to trigger thermal throttling is 80 °C and during thermal throttling, the CPU stalls its current execution. In all experiments, unless otherwise specified, the parameters of the DTB-M policy are set as the following: $nt_{\text{diff}} = 2, t_{\text{slice}} = 100$ ms, $T_{\text{diff}} = 10\,^{\circ}\text{C}, \delta = 0.5\,^{\circ}\text{C}$.

The following criteria are considered to measure the quality of a thermal management policy:

- *Hotspot*: The time spent above a temperature threshold which is 80 °C in our case.
- FT: The finish time of the last task in the system. This criterion measures the performance in a system with static workload.

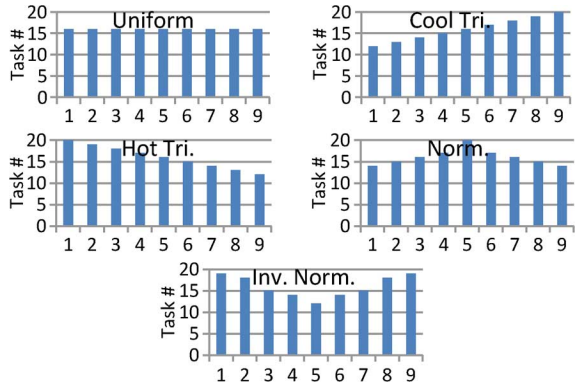


Fig. 11. Different task set generation probability distribution.

- NT: The number of tasks completed within a given period of time. This criterion measures the performance in a system with dynamic workload.
- Mig: Total number of migrations occurred during execution. This criterion measures the migration overhead.

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. Because each invocation of a benchmark program runs only on a single core, its power consumption can be obtained using conventional single processor power estimation tool. We collected their power traces by using the Wattch power analysis tool [4]. The average power consumptions and steady-state temperatures of each task are summarized in Table III. The workloads of the following experiments are random combinations of multiple copies of these nine benchmarks. All experiment results reported below are the average of 10 runs.

A. Workloads Generation

We used both static and dynamic workload in our experiments to evaluate the performance of the DTB-M algorithm. For static workload, each task set is a randomly mixture of 144 CPU benchmarks which are initially distributed evenly across all the 36 PEs. Each PE has four tasks. The number of each benchmark in the task set follows a specific discrete probability distribution of its average power consumption. Fig. 11 shows the five different distributions tested in the experiment.

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; while inverse normal distribution generates more high power tasks and low power tasks than the medium power tasks.

Unlike the static workload, where all tasks are ready from the beginning of the simulation and all of them have the same execution time, with dynamic workload, tasks can be randomly generated on each PE and their execution time follows random distribution. The initial task set is a set of uniformly distributed

TABLE IV
COMPARISON BETWEEN ARMAX AND NEURAL NETWORK MODEL

Workload Distrib.	Predictor	Uni.	Tri. (cool)	Tri. (hot)	Norm.	Inv. Norm.
Hotspot	NN	4986.1	940.2	14223.7	5535.7	4713.1
	ARMAX	8106.4	1722.5	16569	7886.1	7808.7
	%Impr.	38.49	45.42	14.15	29.80	39.64
FT	NN	38140.2	36985.4	40450	38369.7	38053.1
	ARMAX	38392.9	37779	40559.2	38514.6	38453.8
	%Impr.	0.66	2.10	0.27	0.38	1.04
# of Mig.	NN	453.2	242.6	466	406.5	402
	ARMAX	519	556.8	402.8	501	525.7
	%Impr.	12.68	56.43	-15.69	18.86	23.53

CPU benchmarks generated as described above. In every execution interval, a new task is generated on a PE with 0.02 probabilities.

B. Comparison Between Neural Network and ARMAX Predictor Under Static Workload

In the first set of experiments, we investigate the impact of the temperature predictor on the performance of DTB-M policy under static workload. Table IV shows the comparison between the performances of the DTB-M with the neural network model and the ARMAX model for different task distributions.

We can see that with the neural network predictor, the DTB-M is able to reduce the hotspot by 33.5% on average comparing to the ARMAX model. This is because the neural network predictor makes more accurate prediction for the thermal impact of the workload pattern that will be generated after task migration, and helps both the master and slave agents to make thermal safe decisions.

The prediction accuracy also affects the number of migrations. Because using the neural network predictor maintains a more balanced thermal profile and reduces hotspots, the migration request is triggered less often than the system using the ARMAX predictor. On average, the neural network predictor could reduce migration overhead by 19.16%. Note that the ARMAX has less number of migrations in hot triangular distributed workload compared to other four cases. This is because the ARMAX model adapts to the high power tasks and high temperature, and tends to give conservative temperature prediction.

Also note that although the neural network predictor produces much less hotspot, and invokes thermal throttling less frequently, it does not improve the finish time of the system a lot. This is due to two reasons. First, the thermal throttling time is much shorter compared to the task execution time. Second, the finish time is determined by the last task completed by the PE which invokes the most thermal throttling. The worst case numbers of thermal throttling in a system using the neural network predictor and the ARMAX predictor are about the same.

In this experiment, we did not show the comparisons of thermal gradients and thermal cycles. Because both systems have high CPU utilizations and both of them have low thermal gradients and thermal cycles.

C. Comparison between Dynamic Workload

In the second set of experiments, we compare the impact of different temperature predictors on the performance of the DTB-M policy under dynamic workload. The execution time of the task is uniformly distributed between 15 to 30 execution intervals, which is equivalent to 1.5 to 3 s. We simulate both systems for equal length of period and compare their performance.

TABLE V
PERFORMANCE WITH DYNAMIC WORKLOADS

Predictor	Hotspot	NT	# of Mig.
ARMAX	16025.5	196.5	1048.5
Neural Network	11230.3	202.9	806.5
Improvement (%)	29.92	3.26	23.08

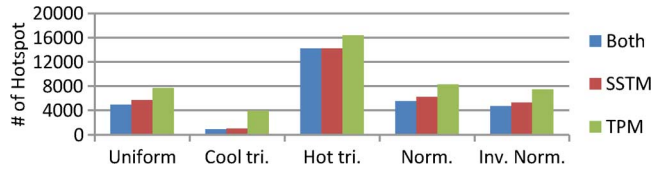


Fig. 12. Comparison of hotspots.

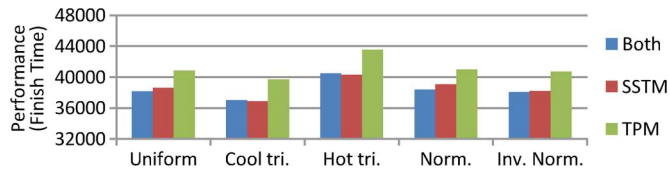


Fig. 13. Comparison of performance.

As shown in Table V, compared to the DTB-M with ARMAX predictor, the DTB-M with neural network predictor improves the system performance by 3.26%, reduces the hotspots by 29.92% with 23.08% less migration overhead. Note that under the dynamic workload, we can see more system performance improvement as the result of using a better temperature predictor than that with the static workload. This is because the number of tasks is not fixed in dynamic workload. The less thermal throttling occurs, the more time could be used for tasks and hence more tasks are completed. While with static workload, the performance is determined by the PE who spends the longest time in thermal throttling.

D. Comparison Between SSTM and TPM Policies

In the third set of experiments, we evaluate the individual performance of the SSTM and TPM policy.

Figs. 12–14 shows the hotspot, performance and migration overhead of the combined migration scheme (i.e., DTB-M) as well as those for the individual SSTM and TPM migrations. As we could expect, the migration overhead for using the combined scheme is larger than employing any one of these two schemes individually. However, it is less than the sum of those individual schemes. This is because the migration decisions made by SSTM and TPM are not mutually exclusive.

Fig. 12 shows that the DTB-M reduces hotspots by 9.12% over SSTM and 39.06% over TPM on average. We can see that the SSTM is more effective in hotspot reduction compared to the TPM. This is because the SSTM reduces hotspots by mapping tasks according to the PE's heat dissipation ability while TPM reduces hotspots by interleaving high power tasks with low power tasks on the same PE. The PE's own heat dissipation ability plays a more important role in preventing the high temperature than interleaving low power tasks and high power tasks on the same PE. Because the SSTM is more effective in

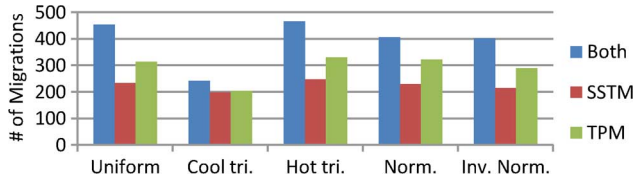


Fig. 14. Comparison of number of migrations.

TABLE VI
COMPARISON BETWEEN GLOBAL AND DISTRIBUTED POLICY

Workload Distribution	Policy	Uniform	Cool tri.	Hot tri.	Norm.	Inv. Norm.
Hotspot	Distributed	4986.1	940.2	14223.7	5535.7	4713.1
	Global	4343	783.5	13943.7	4737.4	4176.4
	%Impr.	14.81%	20.00%	2.01%	16.85%	12.85%
FT	Distributed	38140.2	36985.4	40450	38369.7	38053.1
	Global	37375.4	36501.7	38662.5	37534.7	37334.3
	%Impr.	2.05%	1.33%	4.62%	2.22%	1.93%
# of Mig	Distributed	453.2	242.6	466	406.5	402
	Global	391.9	273.7	508.5	384.3	386.8
	%Impr.	15.64%	-11.36%	-8.36%	5.78%	3.93%

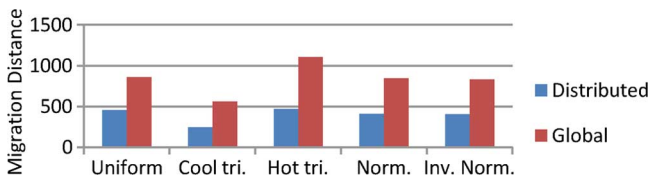


Fig. 15. Comparison of migration distance.

hotspot reduction than TPM, in DTB-M, we give SSTM higher priority and perform it before the TPM.

E. Comparison Between Distributed Policy and Global Policy

As we mentioned at the beginning of the paper, distributed thermal management has lower control and monitoring overhead than centralized thermal management. However, it also has limitations such as low convergence speed, sub-optimal solutions, etc. In the fourth set of experiments, we compare the DTB-M policy with a global version of the same migration scheme to assess the significance of these limitations.

The global policy performs the same DTB-M migration with the assumption that all PEs on the same chip are the nearest neighbor to each other, therefore, task migration could happen between any two PEs. The experiment assumes that there is a central controller in the system and it controls the task exchange and migration between any PEs. The experiment also assumes that all information exchange between PEs and the controller take the same amount of time. This gives a bias to the global policy whose communication time actually should be longer due to multi-hop communication path.

Table VI shows the comparison between DTB-M policy and the global policy in terms of the number of hotspots, system performance and the number of migrations under static workload. In average, the global policy reduces the hotspots by 13.3%, finishes all tasks 2.43% faster and has 1.13% less number of migrations compared to the distributed policy. It is not surprising that the global policy outperforms the distributed policy in all aspects, because it can move the task to a better position more quickly.

TABLE VII
COMPARISON BETWEEN GLOBAL AND DISTRIBUTED POLICY UNDER EXTREME CASES

Workload Distribution	Policy	Uniform	Cool tri.	Hot tri.	Norm.	Inv. Norm.
Hotspot	Dist. Corner	18088	12152.1	25106.9	17570.1	19107.9
	Glob. Corner	3659.7	554.3	13239.3	4203.8	3915.6
	Dist. Center	9239.4	3854.6	14931.4	10198.5	7246.3
	Glob. Center	3713.4	908.6	13167.9	4520.5	3848.2
FT	Dist. Corner	43869.2	41064.1	45404.6	44510	44063.4
	Glob. Corner	37270.8	36477	39235.9	37744.4	37408.6
	Dist. Center	41280.8	38400	44792.4	43483.1	38942.6
	Glob. Center	37445.5	36472.8	39178.8	38050.3	37288.2
Migration Distance	Dist. Corner	69.25	55.85	83.95	75.4	64.75
	Glob. Corner	813.8	527.1	1027.8	743.7	834.1
	Dist. Center	193.5	143.2	182.1	160.8	193.5
	Glob. Center	861.3	622.9	1144.1	862	861.3

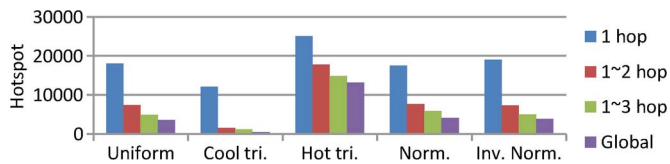


Fig. 16. Comparison of hotspots between multi-hops distributed policy and global policy.

In spite of the hotspot reduction and performance improvements, one major drawback of the global policy is that, since the tasks are exchanged globally, its migration distance is much longer than that of the distributed policy. Fig. 15 shows the comparison of the total migration distance between the global policy and the distributed policy. The migration distance of the global policy is 2.14 times longer than that of the distributed policy on average. Overall, the performance improvement of the global policy is not as significant as the increase of the overhead.

In the previous experiment, the initial task mapping is randomly generated. Therefore, the high power tasks and low power tasks are evenly distributed across the system. In next experiment, we test two extreme cases, both of which have high concentration of high power tasks in a small area in the initial mapping. The further a PE is away from this area, the higher probability that it will be assigned to a low power task. In the first test case, the "hot area" is located at the corner of the chip, while in the second test case it is located at the center of the chip.

Table VII presents the performance of the global policy and the distributed policy for the two extreme cases. Comparing the results in Table VI and Table VII, we can see that the performance of the global policy is hardly 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 these two extreme cases. It is because the distributed policy relies on a rippling process to pass out high power tasks and it can be very slow.

In order to speed up the rippling process, we slightly modify the distributed policy by allowing a PE to send migration requests to its far neighbors occasionally. Figs. 16–18 show the performance of the original distributed DTB-M policy where communication only happens between 1-hop neighbors, the modified DTB-M policy where communication could happen between 1- and 2-hop neighbors and the modified DTB-M

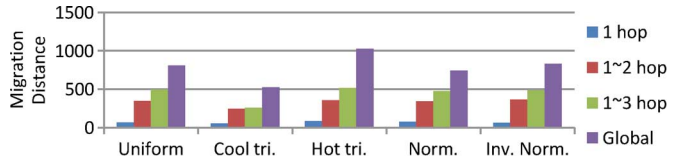


Fig. 17. Comparison of migration distance between multi-hops distributed policy and global policy.

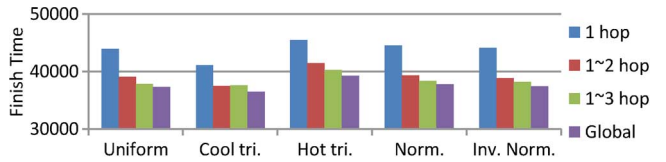


Fig. 18. Comparison of finish time between multi-hops distributed policy and global policy.

TABLE VIII
COMPARISON BETWEEN DTB-M, PTB, AND PTB-NN

Workload Distribution	Policy	Uniform	Cool tri.	Hot tri.	Norm.	Inv. Norm.
Hotspot	DTB-M	1204.5	212.3	5496.9	1455.3	1079.4
	PTB	2355.3	184.8	7318.6	2053.1	2769.5
	%Impr.	48.86%	-14.88%	24.89%	29.12%	61.03%
	PTB-NN	1394.1	157.2	6249.4	1430.5	1699.7
	%Impr.	15.74%	-25.95%	13.69%	-1.70%	57.47%
FT	DTB-M	10655.5	9632.4	12820.9	10922.2	10595.8
	PTB	11166.3	9252.5	11725.6	10800.3	11167.2
	%Impr.	4.57%	-4.11%	-9.34%	-1.13%	5.12%
	PTB-NN	10445.3	9219.1	12137.1	10709.7	10639.3
	%Impr.	-1.97%	-4.29%	-5.33%	-1.95%	0.41%
# of Mig	DTB-M	43.4	18.2	49.4	41.4	44.6
	PTB	265.8	54.2	464.6	189.3	314.3
	%Impr.	83.67%	66.42%	89.37%	78.13%	85.81%
	PTB-NN	129.7	25.2	351.8	116.4	142.4
	%Impr.	66.54%	27.78%	85.96%	64.43%	68.68%

policy where communication could happen between 1-, 2-, and 3-hop neighbors. In the modified DTB-M, the 2-hop and 3-hop neighbors will be contacted with 30% and 10% probability, respectively. As we can see, the percentage difference of hotspots between the distributed and global policies reduces from 75.61% to 28.98% when we extend the communication range from 1-hop neighbors to 2- or 3-hop neighbors; and the percentage difference of finish time between global and distributed policies reduced from 14.02% to 2.06%. Because the far neighbors are contacted with low probability, the total migration distance of the modified DTB-M is still much lower than that of the global policy. As shown in Fig. 17, even if the 2- or 3-hop neighbor are contacted, the migration overhead is still 43% less than that of the global policy.

Finally, we want to point out that in a real system, such extreme cases of initial task allocation rarely happen because it can be avoided by simple random mapping of the tasks.

F. Comparison Between PTB and DTB-M

In this set of experiments, we compare the DTB-M policy with the PTB policy proposed in [9]. PTB reduces hotspots by proactively exchanging tasks between a core which is predicted to be hot and a core which is predicted to be cool. Note that the PTB is not a distributed policy and those two cores do not have to be the nearest neighbors. To obtain a fair comparison, we duplicate the experiment settings in [9] and assign only one task to each core. The original PTB policy employs AMAR model for temperature prediction. To separate the disturbance from different temperature prediction models, we replace the AMAR model in the PTB policy with our neural network model and name it PTB-NN.

Table VIII shows the comparison among the DTB-M policy, the original PTB policy and the PTB-NN policy. Compared to the PTB and PTB-NN policies, the DTB-M policy successfully reduces the hotspots by 29.8% and 11.85%, reduces migration overhead by 80.68% and 62.68%, while only has 0.98% and 2.63% performance degradation on average, respectively. This is because a PE using DTB-M policy always analyzes the workload before offering task exchange. If the task migration will not be benefit or, even worse, will cause hotspots, it will not be performed. However, such analysis does not take place in PTB and PTB-NN. Because the DTB-M does not perform any unnecessary migrations, its migration overhead is also lower. Note that the PTB and PTB-NN policy are global policies; the thermal throttling time is more evenly distributed among all cores than DTB-M. Thus the performance is slightly better than that of the DTB-M policy for some test cases.

VIII. CONCLUSION

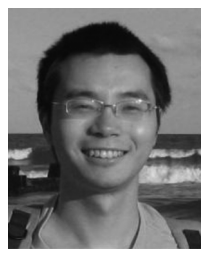
In this paper, we proposed a distributed thermal management framework for many-core systems. In this framework, no centralized controller is needed. Each core has an agent which monitors the core temperature, communicates and negotiates with neighboring agents to migrate and distribute tasks evenly across the system. The agents use DTB-M policy for task migration. Our DTB-M policy consists of two parts. The SSTM migration policy distributes different tasks in a neighborhood based on their heat dissipation ability. The TPM migration policy ensures a good mixture of hot tasks and cool tasks on processors in a neighborhood. We also proposed a neural network-based prediction model that can be used not only for future temperature prediction but also for agents to evaluate the rewards of proposed migration offers.

We compared our neural network predictor with an extended version of ARMA predictor and showed that our predictor can make prediction faster and more accurate in the system where tasks start, complete and migrate dynamically. We showed that our DTB-M policy reduces 29.8% hotspots and 80.68% migration overhead with only 0.98% performance overhead compare to the PTB thermal management.

REFERENCES

[1] Tilera Corporation, San Jose, CA, "Tile processor architecture: Technology brief," 2008. [Online]. Available: http://www.tilera.com/pdf/ProductBrief_TileArchitecture_Web_v4.pdf
[2] I. Aleksander and H. Morton, *An Introduction to Neural Computing*. London, U.K.: International Thomson Computer Press, 1995.
[3] S. Borkar, "Thousand core chips—A technology perspective," in *Proc. Design Autom. Conf. (DAC)*, 2007, pp. 746–749.
[4] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural level power analysis and optimizations," in *Proc. Int. Symp. Comput. Arch. (ISCA)*, 2000, pp. 83–94.
[5] X. Chen, C. Xu, R. Dick, and Z. Mao, "Performance and power modeling in a multi-programmed multi-core environment," in *Proc. Design Autom. Conf. (DAC)*, 2010, pp. 813–818.
[6] J. Choi, C. Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, "Thermal aware task scheduling at the system software level," in *Proc. Int. Symp. Low Power Electron. Design*, 2007, pp. 213–218.

[7] R. Cochran and S. Reda, "Consistent runtime thermal prediction and control through workload phase detection," in *Proc. Design Autom. Conf. (DAC)*, 2010, pp. 62–67.
[8] A. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proc. Design Autom. Test Euro. (DATE)*, 2007, pp. 1659–1664.
[9] A. Coskun, T. Rosing, and K. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1503–1516, Oct. 2009.
[10] W. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Design Autom. Conf.*, 2001, pp. 684–689.
[11] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *Proc. Int. Symp. Comput. Arch. (ISCA)*, 2006, pp. 78–88.
[12] T. Ebi, M. Al Faruque, and J. Henkel, "TAPE: Thermal-aware agent-based power economy for multi/many-core architectures," in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2009, pp. 302–309.
[13] 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," in *Proc. Int. Solid-State Circuits Conf. (ISSCC)*, 2010, pp. 108–109.
[14] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proc. Asia South Pacific Design Autom. Conf. (ASP-DAC)*, 2003, pp. 233–239.
[15] W. Huang, M. Stan, K. Sankaranarayanan, R. Ribando, and K. Skadron, "Many-core design from a thermal perspective," in *Proc. Design Autom. Conf. (DAC)*, 2008, pp. 746–749.
[16] R. Jayaseelan and T. Mitra, "Dynamic thermal management via architectural adaption," in *Proc. Design Autom. Conf. (DAC)*, 2009, pp. 484–489.
[17] L. Ljung, *System Identification: Theory for the User (2nd Edition)*. Upper Saddle River, NJ: Prentice-Hal PTR, 1999.
[18] 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.*, vol. 28, no. 1, pp. 3–21, Jan. 2009.
[19] P. Michaud, A. Seznec, D. Fetis, Y. Sazeides, and T. Constantinou, "A study of thread migration in temperature-constrained multicores," *ACM Trans. Arch. Code Optim. (TACO)*, vol. 4, no. 2, pp. 9-1–9-28, Jun. 2007.
[20] D. Wentzlaff, C. Gruenwald, N. Beckmann, K. Modzelewski, A. Belay, L. Youseff, J. Miller, and A. Agarwal, "A Unified operating system for clouds and manycore: FOS," MIT-CSAIL-TR-2009-059, Nov. 2009.
[21] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, D. Atienza, and G. De Micheli, "Thermal balancing policy for streaming computing on multiprocessor architectures," in *Proc. Design Autom. Test Euro. (DATE)*, 2008, pp. 734–739.
[22] S. Murali, A. Mutapcic, D. Atienza, R. Gupta, S. Boyd, and G. De Micheli, "Temperature-aware processor frequency assignment for MPSoCs using convex optimization," in *Proc. CODES+ISSS*, Sep. 2007, pp. 111–116.
[23] V. Nollet, T. Marescaux, and D. Verkest, "Operating system controlled network on chip," in *Proc. Design Autom. Conf. (DAC)*, 2004, pp. 256–259.
[24] R. Rao and S. Vrudhula, "Fast and accurate prediction of the steady state throughput of multi-core processors under thermal constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1559–1572, Oct. 2009.
[25] L. Shang, L. Peh, A. Kumar, and N. Jha, "Thermal modeling, characterization and management of on-chip networks," in *Proc. Int. Symp. Microarch.*, 2004, pp. 67–78.
[26] S. Sharifi, A. Coskun, and T. Rosing, "Hybrid dynamic energy and thermal management in heterogeneous embedded multiprocessor," in *Proc. Asia South Pacific Design Autom. Conf. (ASPDAC)*, 2010, pp. 873–878.
[27] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Trans. Arch. Code Opt.*, vol. 1, no. 1, pp. 94–125, Mar. 2004.
[28] 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," in *Proc. Int. Solid-State Circuits Conf. (ISSCC)*, 2007, pp. 98–589.
[29] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *Proc. Int. Symp. Comput. Arch. (ISCA)*, 2009, pp. 314–324.
[30] J. Wawrzynek, D. Patterson, M. Oskin, S. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic, "RAMP: Research accelerator for multiple processors," *IEEE Micro*, vol. 27, no. 2, pp. 46–57, Aug. 2007.
[31] I. Yeo, C. Liu, and E. Kim, "Predictive dynamic thermal management for multicore systems," in *Proc. Design Autom. Conf. (DAC)*, 2008, pp. 734–739.
[32] F. Zanini, D. Atienza, and G. De Micheli, "A control theory approach for thermal balancing of MPSoC," in *Proc. Asia South Pacific Design Autom. Conf. (ASPDAC)*, 2009, pp. 37–42.
[33] S. Liu, J. zhang, Q. Wu, and Q. Qiu, "Thermal-aware job allocation and scheduling for three dimensional chip multiprocessor," in *Proc. Int. Symp. Quality Electron. Design (ISQED)*, 2010, pp. 390–398.
[34] Y. Ge, P. Malani, and Q. Qiu, "Distributed task migration for thermal management in many-core systems," in *Proc. Design Autom. Conf. (DAC)*, 2010, pp. 579–584.



Yang Ge (S'10) received the B.S. degree in telecommunication engineering from Zhejiang University, Hangzhou, China, in 2007, and the M.S. degree from the Department of Electrical and Computer Engineering, Binghamton University, Vestal, NY, in 2009. He is currently pursuing the Ph.D. degree from the Department of Electrical Engineering and Computer Science, Syracuse University, Syracuse, NY.

His research interests include power and thermal analysis and optimization for multi and many-core system.

Qinru Qiu (M'01) received the B.S. degree from the Department of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China, in 1994, and the M.S. and Ph.D. degrees from the Department of Electrical Engineering, University of Southern California, Los Angeles, in 1998 and 2001, respectively.

She is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, Syracuse University. Before joining Syracuse University, she was an Assistant Professor and then an Associate Professor with the Department of Electrical and Computer Engineering, State University of New York, Binghamton. Her research areas include energy efficient computing systems, energy harvesting real-time embedded systems, and neuromorphic computing. She has published over 40 research papers in referred journals and conferences. Her works are supported by NSF, DoD and Air Force Research Laboratory.

Qing Wu (M'01) received the B.S. and M.S. degrees from the Department of Information Science and Electronics Engineering, Zhejiang University, Hangzhou, China, in 1993 and 1995, respectively, and the Ph.D. degree in electrical engineering from the Department of Electrical Engineering—Systems, University of Southern California, Los Angeles, in 2001.

He is currently with the Information Directorate of United States Air Force Research Laboratory, Rome, NY. His research interests include neuromorphic computing algorithms and architectures, hardware/software optimization for massively parallel high-performance computing systems, low power design and power management for cloud computing and green data centers, low power design methodologies for energy harvesting mobile computing systems, power estimation of VLSI circuits and systems, FPGA-based hardware-accelerated computing.