12-2012

Dynamic Thermal Management for Microprocessors

Yang Ge
Syracuse University

Abstract

In deep submicron era, thermal hot spots and large temperature gradients significantly impact system reliability, performance, cost and leakage power. Dynamic thermal management techniques are designed to tackle the problems and control the chip temperature as well as power consumption. They refer to those techniques which enable the chip to autonomously modify the task execution and power dissipation characteristics so that lower-cost cooling solutions could be adopted while still guaranteeing safe temperature regulation. As long as the temperature is regulated, the system reliability can be improved, leakage power can be reduced and cooling system lifetime can be extended significantly.

Multimedia applications are expected to form the largest portion of workload in general purpose PC and portable devices. The ever-increasing computation intensity of multimedia applications elevates the processor temperature and consequently impairs the reliability and performance of the system. In this thesis, we propose to perform dynamic thermal management using reinforcement learning algorithm for multimedia applications. The presented learning model does not need any prior knowledge of the workload information or the system thermal and power characteristics. It learns the temperature change and workload switching patterns by observing the temperature sensor and event counters on the processor, and finds the management policy that provides good performance-thermal tradeoff during the runtime.

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 thesis, we present 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 presented 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.

Temperature affects the leakage power and cooling power. In this thesis, we address the impact of task allocation on a processor's leakage power and cooling fan power. Although the leakage power is determined by the average die temperature and the fan power is determined by the peak temperature, our analysis shows that the overall power can be minimized if a task allocation with minimum peak temperature is adopted together with an intelligent fan speed adjustment technique that finds the optimal tradeoff between fan power and leakage power. We further present a multi-agent distributed task migration technique that searches for the best task allocation during runtime. By choosing only those migration requests that will result chip maximum temperature reduction, the presented framework achieves large fan power savings as well as overall power reduction.

Dynamic Thermal Management for Microprocessors

by

Yang Ge

B.S., Zhejiang University, China, 2007

M.S., Binghamton University, 2009

Dissertation

Submitted in partial fulfillment of the requirements for the degree

Doctor of Philosophy in Electrical and Computer Engineering

Syracuse University

December 2012

To my parents Haitao Ge, Mingyuan Chen, my wife Yukan and my daughter Shannon

Acknowledgement

First of all, I would like to express my deep appreciation and gratitude to my advisor, Dr. Qinru Qiu, for her patient guidance and mentorship she provided to me, all the way from when I was attending her first VLSI class through to the completion of this degree. I am truly fortunate to have had the opportunity to work with her. I would like to thank my committee members, Dr. Roger Chen, Dr. Ezzat Khalifa, Dr. Jae Oh, Dr. Jian Tang and Dr. Senem Velipasalar for their valuable feedback. I would also like to thank Dr. Qing Wu for his guidance and advices.

I would like to thank my fellow doctoral students—those who have graduated, those in the quagmire, and those just beginning—for the discussion, feedback, and friendship.

I have been surrounded with an exceptional circle of friends, and would like to thank Yifan Xu, Yuanyuan Wang, Yu Jiang and Xian Li for their support and the great time we have spent together. Without them, I would not have so much fun in my PhD life.

I would like to thank my cousin Shushu and Yinyin who give me so much help and support since the first day I arrived United States. It has been always exiting to visit them during the breaks and holidays.

The greatest thanks are for all my family, who loved, supported and motivated me. I feel extremely lucky to have such a wonderful family. I am dedicating this thesis to my parents, Haitao Ge and Mingyuan Chen, and my wife Yukan Zhang. My parents have taught me to believe in myself and made me the person I am today. My wife always encourages me during the stressful time and shoulders all the household burdens while I pursued my PhD degree. Without their love and support, I would not be able to come this far.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

Moore's law states that the number of transistors on a chip doubles about every two years or less. 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 [13]. Examples of such system are the 80 tile network-on-chip that has been fabricated and tested by Intel [60], their 48 core single chip cloud computer [28] and Tilera's 64 core TILE64 processor [8]. While the muti-core or many-core technology delivers extraordinary performance, they have to face the significant power and thermal challenges. This is because as we continue to shrink the chip sizes and extract the performance of our systems at the cost of higher power consumption, the ever-increasing chip complexity and power density elevate peak temperatures of the chip and imbalance the thermal gradients. Raised peak temperatures reduce the life-time of the chip, deteriorate its performance, affect the reliability and increase the cooling cost [54]. The adverse positive feedback between leakage power and raised temperature creates the potential of thermal runaway. When mapped on a multi or many-core system, the diverse workload of applications may lead to power and temperature imbalance among different cores. Such temporal and spatial variation in temperature creates local temperature maxima on the chip called the hotspot [24]. An excessive spatial temperature variation, which is also referred to as the thermal gradient, increases clock skews and decreases performance and reliability. Elevated temperatures require more cooling efforts; to cool down the processor, a typical cooling fan can consume up to 51% power budget of a server [35][11].

1.1 The Adverse Effects of Unmanaged Temperature

1.1.1 The Effects of Temperature on System Reliability

One of the most obvious results of high power consumption is the elevated chip temperature, and the most serious consequence of high temperature is the harm to the system reliability. The followings are some common temperature related semi-conductor devices failure mechanism [3]:

- *Electromigration* (*EM*): The deformation of the metal wire caused by the gradual movement of the ions in the metal wire due to the momentum transfer between conducting electrons and diffusing metal atoms. It could eventually break the metal wire and lead to device failure.

- *Stress Migration* (*SM*): The deformation of the metal wire caused by the movement of metal atoms under the influence of mechanical stress gradients. The stress could be generated by differing thermal expansion rates of the materials. It could also break the metal wire and lead to device failure.

- *Dielectric Breakdown* (*DB*): The dielectric fails when a conductive path forms in the dielectric, shorting the anode and cathode in the circuit.

The failure rate *Time-to-Failure* (*TF*) due to these mechanisms could be expressed in the form of Arrhenius equation:

$$TF = A \cdot e^{E_a/(k \cdot T)} \tag{1.1}$$

Where A is a constant, E_a is the activation energy in electronvolts (eV) and k is the Boltzmann's constant (8.62×10^{-5} eV/K). They are all positive constants. T is the operating temperature of the device in Kelvin. Therefore, TF is a decreasing function of the temperature.

When the temperature increases, the TF will decrease exponentially. This means the device could fail more quickly.

High temperature is not the only cause of system reliability issues. In addition, two other thermal phenomena, i.e. thermal cycling and thermal gradients, could also affect the reliability of the device severely. Thermal cycling means the temporal fluctuation of the operating temperature, and it could be caused by, for example, the normal power-up and power-down operation of the device. Other major sources of thermal cycling are from the variation of workload as well as the power management policy of the device, which could potentially switch the components of the device between several power modes very frequently. Thermal cycling could weaken the materials and cause different types of failures like dielectric cracking and solder fatigue etc. The failure rate due to thermal cycling could be expressed in the form of equation (1.2):

$$\lambda \propto (\Delta T)^q \tag{1.2}$$

Where ΔT is the magnitude of the thermal cycle, while q is the frequency of the thermal cycle. The larger and the more frequent of the temperature swing, the bigger the failure rate is.

Thermal gradient is the spatial imbalance of the temperature across the chip. It could be caused by the workload imbalance on today's System-on-Chip (SoC) or the multi/many-core chip. The major problem caused by large thermal gradient is the imbalance of the interconnect resistance, which could lead to increased clock skew. This is very undesirable in synchronized digital circuits and could eventually result in timing violations.

1.1.2 The Effects of Temperature on Leakage Power

Temperature is not only the consequence of the power; they actually interact as both cause and effect to some degree. This is because the leakage power consumption strongly depends on the operating temperature. According to ITRS roadmap [4], leakage power, which already contributes a significant portion of the total system power in current process, will continue to grow as the technology scales as shown in Figure 1.1.

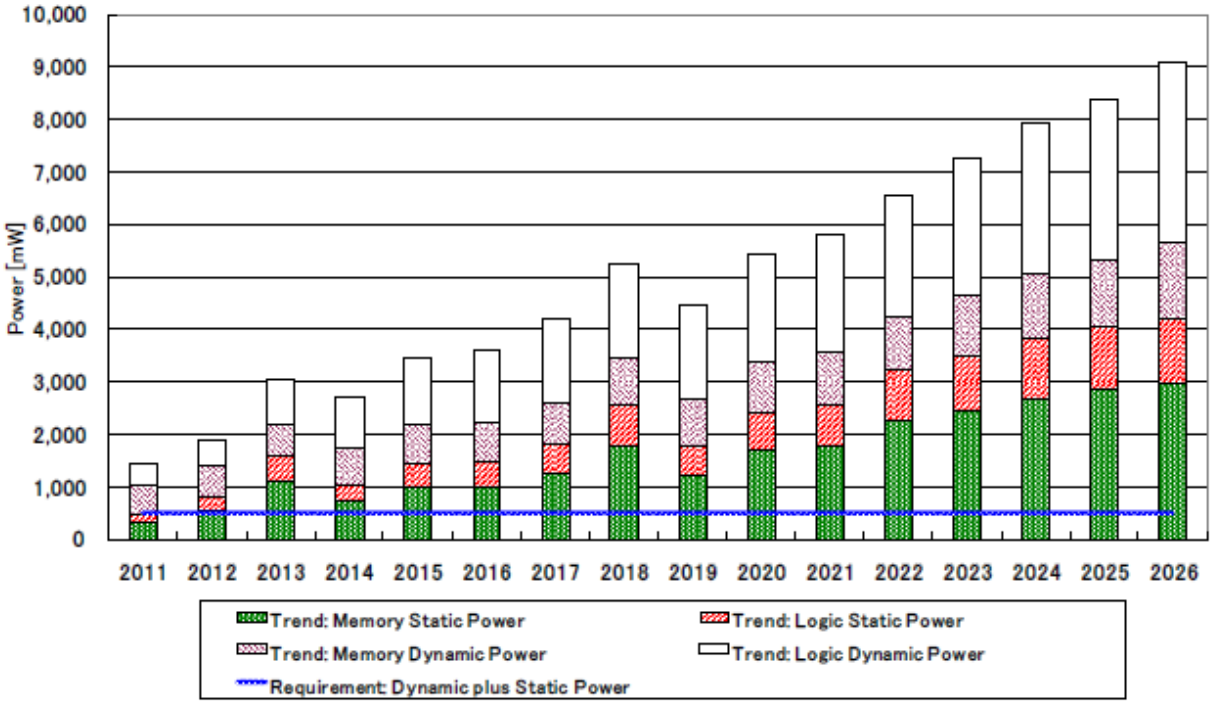


Figure 1.1 System-on-Chip (SOC) Consumer Portable Power Consumption Trends [4]

Previous works have investigated the dependence between temperature and leakage power extensively [36][66]. And the leakage current could be formulated as:

$$I_{leak} = I_s \cdot A \cdot T^2 e^{\frac{\alpha+\beta V_{dd}}{T}} + B$$

Where A, B, α and β are positive, temperature-independent constants for a given technology, V_{dd} is the supply voltage and I_s is the reference leakage current for some given

temperature and supply voltage. We could see that the temperature scaling for the leakage current (and the leakage power) is $T^2 e^{1/T}$. Figure 1.2 shows the leakage current against the temperature curve.

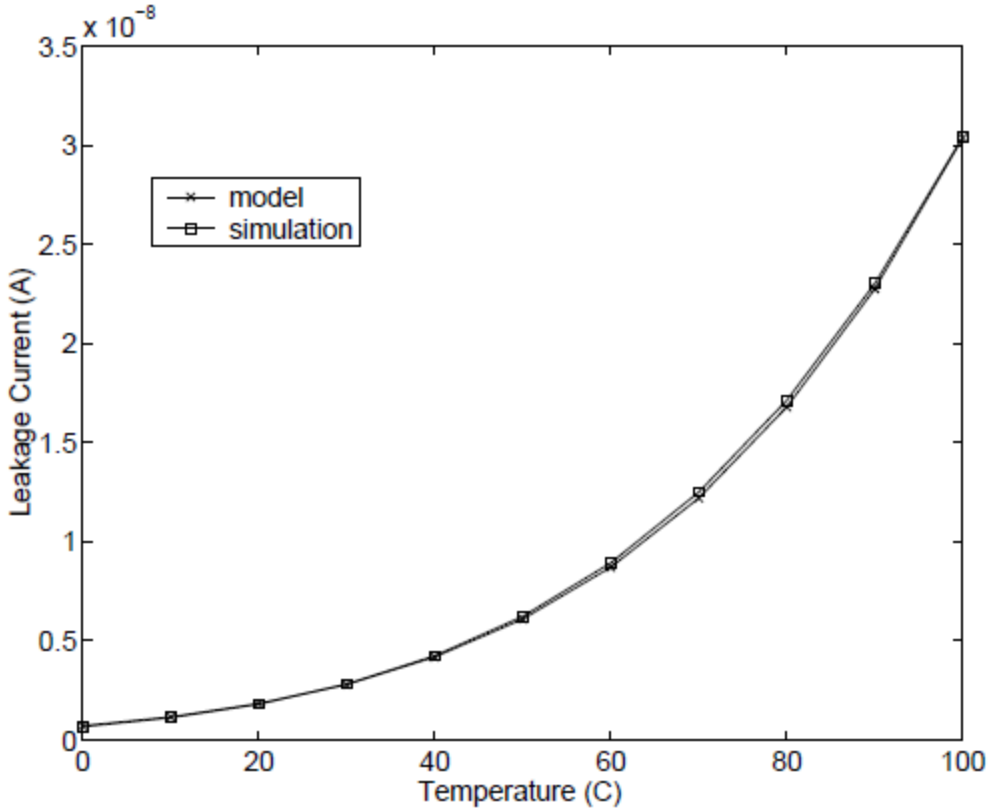


Figure 1.2 Leakage Current v.s. Temperature Relation[66]

1.1.3 The Effects of Temperature on Cooling System

Another obvious consequence of elevated temperature is the higher cooling cost. To guarantee that the chip operates normally, the chip has to be equipped with more expensive packaging. Or for a conventional heat-sink-based cooling solution, the cooling fan has to operate at higher speed to accommodate the extra heat generated by the chip. This is because the heat dissipation ability of the heat sink is determined by its thermal resistance R_{h2a} (h2a stands for heat sink to ambient) which is in turn determined by the cooling fan speed as shown in (1.3) [11].

$$R_{h2a} = (h_1 v_f (1 - e^{\frac{h_2 v_f^{h_3} + h_4}{h_1 v_f}}))^{-1} \quad (1.3)$$

In this equation h_1, h_2, h_3 and h_4 are chip specific physical coefficients, and v_f is the fan speed. And the fan power consumption P_{fan} is proportional to the cubic of the fan speed, i.e. $P_{fan} \propto v_f^3$. This means high temperature not only raises the fan power consumption, but also reduces fan life time.

1.2 Dynamic Thermal Management Techniques

We have seen in the previous section that unmanaged temperature could lead to serious reliability, performance, power as well as cost issues. *Dynamic Thermal Management* (*DTM*) techniques are designed to tackle the aforementioned problems and control the chip temperature as well as power consumption. As long as the temperature is regulated, the system reliability can be improved. It has been pointed out that a moderate reduction in temperature by 10^{o}C~15^{o}C can extend the lifespan of the electronic device 2 times [34]. And a 10^{o}C decrease in the magnitude of thermal cycles can achieve 16 times increase in mean-time-to-failure for metallic structures. Leakage power also drops significantly when temperature decreases. For every 9^{o}C temperature reduction, there is 50% reduction in the leakage power [38]. This reduction is particularly important in the future System-on-Chip design, because the leakage power consumption is estimated to account for more than 50% of total chip power consumption [4]. Regulated temperature not only guarantees the system reliability and reduces leakage power consumption, but also boosts the performance. Transistor switching speed is faster in low temperature [49]. A balanced spatial gradient can mitigate the clock skew problem noticeably.

Dynamic thermal management refers to those techniques which enable the chip to autonomously modify the task execution and power dissipation characteristics so that lower-cost cooling solutions could be adopted while still guaranteeing safe temperature regulation. A DTM controller observes system information during runtime and takes thermal management actions accordingly. It maintains the system temperature below a safe threshold or reduces thermal violations as much as possible and produces thermal profile that is as smooth as possible with minimum performance overhead.

The most important system information that is required to carry out the dynamic thermal management for a computing system is the chip temperature. This information can be read from the on-chip temperature sensors or estimated using a thermal model. In additional to the temperature information, application characteristics, task power consumptions, etc. are also needed by some state-of-the-art DTM techniques.

1.2.1 The Difference between Dynamic Power Management (DPM) and Dynamic Thermal Management Techniques

Although temperature is essentially the consequence of power consumption, and dynamic thermal management techniques also require modifying the power dissipation characteristics, and furthermore, both DPM and DTM share some actions like *Dynamic Voltage Frequency Scaling* (*DVFS*) and task migration, there are significant differences existing between thermal aware design and power aware design. First of all, the system thermal profile cannot be characterized by its power consumption profile alone. This is because the power consumption could change instantaneously, but the temperature is the cumulative effect of power consumption and changes gradually in time and space domain. To make up an analogy, power consumption acts like the current source in an RC circuit, while the temperature acts like the voltage on each

node. So temperature behaves like a low-pass filter, filtering out the high frequency component of the power consumption variation. Second, temperature is not proportional to power itself, but proportional to the power density, i.e. $T \propto P/A$, where A is the area. Therefore, even if it is impossible to reduce power consumption, but we still want to reduce the temperature, we could distribute the power consumption to a large area. For example, instead of consolidating all the threads onto one core and stressing only a small portion of the chip, it is better to assign them to multiple cores from a thermal perspective. Third, compared to thermal management policy, power management policy could have conflicting goals and potentially produce undesirable thermal profile. For example, a power management policy might switch the device to low power states very frequently to save power. This behavior could produce large and frequent thermal cycles and accelerate the package fatigue as mentioned in section 1.1.1. In order to achieve low power, power management policy might turn off some components and consolidate computation, which could create localized hotspots and large thermal gradients.

1.3 Thesis Contributions

As we could see, dynamic power management techniques are not able to tackle the problem related with unmanaged temperature and dynamic thermal management techniques are indeed necessary for thermal aware design. To perform the thermal management techniques effectively without hurting the system performance requires exploiting the workload characteristics to make intelligent management decisions. This thesis focuses on the design of low-overhead, performance-efficient, learning-based thermal management techniques that target on the computing platforms where workload information is unknown a priori (like the general purpose personal computer or high performance computing servers) and the thermal/power characteristics for different applications or even the different phases in the same application vary

significantly. These uncertainties impose remarkable challenges on the design and evaluation of DTM policy.

The proposed learning based techniques assume there is a learning agent residing in the system (e.g. a delicate hardware unit or a subsystem in the OS), which is responsible for monitoring the system dynamics and taking appropriate thermal management actions. The learning agent always extracts some information from past system dynamics and fits them into a learning model. The learning model can be a prediction model, a reinforcement learning model or some other models that are suitable for the problem under investigation. Based on the given input, the learning model will make predictions about future system dynamics, like temperature, workload or some abstract reward/penalty values. The learning agent then makes control decisions which are most beneficial to the system (e.g. can reduce temperature most or maximize a reward function) based on the prediction. The learning models can be updated based on the outcome of the thermal management control actions.

For applications with high temporal correlation, this work presents a learning agent that intelligently exploits their property and performs DTM actions at appropriate time granularity. For example, most of the multimedia applications such as MPEG movie clips or MP3 files are naturally arranged into frames. The computation load of processing each frame has high temporal correlation. Then the agent performs the learning-based DTM at a single frame granularity.

Most of the existing dynamic thermal management techniques are centralized approaches. They require a controller that monitors the temperature and workload distribution of the entire chip and make global decisions of resource allocation. Such centralized approaches do not have

good scalability. As the number of processing elements grows, the complexity of solving the resource management problem grows super-linearly. Furthermore, a centralized monitoring and commanding framework incurs a large overhead, as communication between central controller and cores will increase exponentially [25].

In this work, we present 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 core. The agent observes the workload and temperature of local processor while communicating and exchanging tasks with its nearest neighbors. The goal of task migration is to distribute tasks to processors based on their heat dissipation capabilities and also ensure that each processor has a good mix of high power (i.e. "hot") tasks and low power (i.e. "cool") tasks.

The contributions of this thesis are outlined as the following:

- This thesis provides the reader an in-depth and detailed survey on the working principles and implementation details of some existing state-of-the-art dynamic thermal management techniques in Chapter 2. This thesis presents these techniques in three categories based on the actions they are: Temperature Aware Task Scheduling, Temperature Aware Dynamic Voltage and Frequency Scaling and Temperature Aware Task migration.

- For multimedia applications, this thesis adopts a reinforcement learning-based algorithm to find the optimum thermal management policy during runtime. We consider the processor's DTM controller as a learning agent and model the rest of the system, e.g., the operating temperature, the hardware and the application

status as the environment. After the learning agent, i.e. the DTM controller, takes an action (i.e., switching to a new operating frequency), it observes the environment and estimates the reward or penalty caused by this action. The agent learns from this experience and tries to improve its future action selections to maximize the reward (or minimize the penalty). The details of how to apply reinforcement learning to dynamic thermal management can be found in Chapter 3.

- For a many-core processor, this thesis presents a distributed thermal management framework where no centralized controller is required. The distributed thermal management agent resides in each processor, monitors its own power and temperature dynamics, communicates and exchanges tasks with its nearest neighbor so that each processor has a good mix of high power tasks and low power tasks. Because the communication and migration cost only happens between the nearest neighbors, the communication cost and migration overhead for each core does not increase when the number of cores in the system increases.

- A neural network based peak temperature predictor is also presented in this thesis. It predicts the future peak temperature based on the workload statistics of the local processor and the maximum and minimum temperatures of the neighbors. Once trained, the neural network predictor has very low computation complexity. Because it takes the workload as one of the input parameters, it can give accurate prediction right after task migration. It can even be used to predict the temperature impact of a migration before the migration actually takes place as long as the power consumption of the task that will be migrated in or out is provided.

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 predictor is part of the thermal management agent in each core. The details of the distributed thermal management policy and the design of the neural network temperature predictor are presented in Chapter 4.

- This thesis also investigates the effects of task mapping on the joint optimization of cooling power and computation power for a many-core platform under a given workload. We show that different task mappings could change the temperature distribution across the system and could 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 power. Hence they require different optimization techniques. The detailed optimization techniques are presented in Chapter 5.

Chapter 2 A Survey of Dynamic Thermal Management Techniques

The thermal management problem aims at finding the optimal resource management policy that could effectively control the peak temperature, the number of hot spots and the thermal gradient of a system on chip. Based on when the optimization takes place, the existing thermal management research can be divided into the following two major categories:

- Offline approach: offline techniques usually target at the application specific embedded system that has predictable workload. They solve the temperature aware resource management problem at design time or compile time.

- Online approach: online techniques target more general platform where workload information is unknown at the design time. They rely on state-of-the-art learning or control techniques to adaptively manage the hardware and software to control temperature.

A great number of different dynamic thermal management actions have been investigated. These actions include clock gating, dynamic voltage frequency scaling, computation migration and hybrid methods which combine two or more techniques mentioned above. Although different techniques use different mechanisms and are applied in different computing

environments, they share the same key idea, that is, to modify the power dissipation characteristics of a computing system for less heat generation and a smoother heat distribution.

The three most commonly used thermal management actions are listed as the following:

- Temperature Aware Scheduling: This technique optimizes the task execution order to reduce temporal variation and peak temperature in the system.
- Dynamic Voltage and Frequency Scaling (DVFS): This technique dynamically stalls the processor or scales the processor's supply voltage and frequency to achieve a lower power density.
- Temperature Aware Task Migration: This technique dynamically adjusts the task mapping to reduce spatial thermal gradients and peak temperature of the system.

In this chapter, we will present one dynamic thermal management technique using each action. We focus on online techniques in this chapter. Unlike offline techniques, online techniques are usually applied to more general platforms such as general purpose personal computers or high performance computing servers is priori unknown and the thermal/power characteristics for different applications vary significantly. These uncertainties impose remarkable challenges on the design and evaluation of DTM policy. The core problem here is how the DTM policy reacts and adapts to the system dynamics (processors, applications and environments) so that the objectives (reliability, performance) can be well accomplished.

2.1 Temperature Aware Task Scheduling

As we mentioned earlier, large temperature gradients and thermal cycles can cause clock skew and reliability issues. The authors of [18] aim at controlling task scheduling to achieve a temporally and spatially uniform temperature distribution.

Many modern operating systems (e.g. Linux 2.6 [40]) for multiprocessors maintain a ready queue for each processor. A task generated on a processor would stay in the queue of the processor for cache affinity. The scheduler would not move the task to another processor's queue arbitrarily for performance consideration. Only when there is an obvious load imbalance among processors, the scheduler will start to move tasks from the heavy loaded processor to light loaded processor. This scheduling scheme is called load balancing. However, this scheme does not take the temperature into consideration. Therefore it might not be able to achieve a balanced temperature profile.

The authors of [18] propose several temperature aware scheduling schemes. The first scheme is referred as the *Coolest,* which sends a ready task to a processor with the lowest temperature. The authors further improve the *Coolest* method by considering the lateral heat transfer between neighboring processors. The second method is referred as *Coolest-FLP*. It sends tasks to those processors which have idle neighboring processors with higher priority.

The third scheme is referred as *Adaptive-Random*. It is a probabilistic policy which considers both the performance benefits of load balancing policy and thermal benefits of *Coolest* policy. The general idea of this policy is to assign a probability value to each processor; when a task is ready to run, it will be sent to a processor based on its probability. The probability of a processor will be updated dynamically based on the following equation (2.1).

$$P_{new} = P_{old} + W \tag{2.1}$$

Where P_{new} and P_{old} are the probability after and before the update, and W is the incremental amount defined in the following way: If a processor's temperature has crossed a temperature threshold T_{thr} in the history window, then W is set to a negative value and $W = -P_{old}$. This means that the probability of this processor drops to 0 and it will not be assigned with new tasks in near future. If the temperature of a processor drops below T_{low}, then W is set to a positive value and $W = T_{avg}/T_{thr}$, where T_{avg} is the average temperature over the history window. This ensures that a processor with lower temperature will have a better chance to receive a new task.

The authors validate the effectiveness of their scheduling policy on an 8-core UltraSPARC [33] based processor model using applications with different characteristics (e.g. CPU intensive, memory intensive and network intensive). The *Adaptive-Random* scheduling policy is able to control the hotspots (i.e., the running time above 85^{o}C) down to 0.09% and the temporal variation (defined as the average temperature change over a window of 1000 seconds) down to 0.15%. This policy also reduces the spatial variation (defined as the percentage of time when there are cores whose temperature difference is more than 20^{o}C) by about 70%.

2.2 Temperature Aware DVFS for CMP Power Control

In [61], the authors address the problem of joint management of chip power consumption and temperature distribution for a chip multiprocessor (CMP). The adopted control action is DVFS; and the control algorithm is based on the Model Predictive Control (MPC) theory, which is a well-established feedback control model with Multiple-input and Multiple-output (MIMO). Feedback control is an effective tool for power and temperature management due to its theoretically guaranteed control accuracy and stability. And MIMO controller is especially useful

for multiprocessor power/thermal management because first, the small overhead of per-core DVFS makes it possible to simultaneously change the DFVS level of multiple cores; second, the significant workload intensity variation and process variation among cores require them to response differently. Figure 2.1 shows the system control loop architecture. At every control step, the power controller collects following information from CMP: the power consumption of the chip, the temperature of each core and the system level performance of each core. The total chip power consumption is the ***controlled variable*** of the control loop. Based on the collected information, the power controller computes the new DVFS level for each core of the CMP and DVFS modulator adjusts the DVFS level accordingly. The DVFS level of each core is the ***manipulated variable*** in the control loop. The power consumption of each core at control step k is modeled as a variable linearly dependent on the operating frequency at step k as shown in equation (2.2) and its differential form is shown in equation (2.3). The total chip power consumption $cp(k)$ is the summation of each core's power consumption (2.4). Note that the $\Delta f_i(k)$ is the manipulated variable of the control loop and should be computed by the MPC controller.

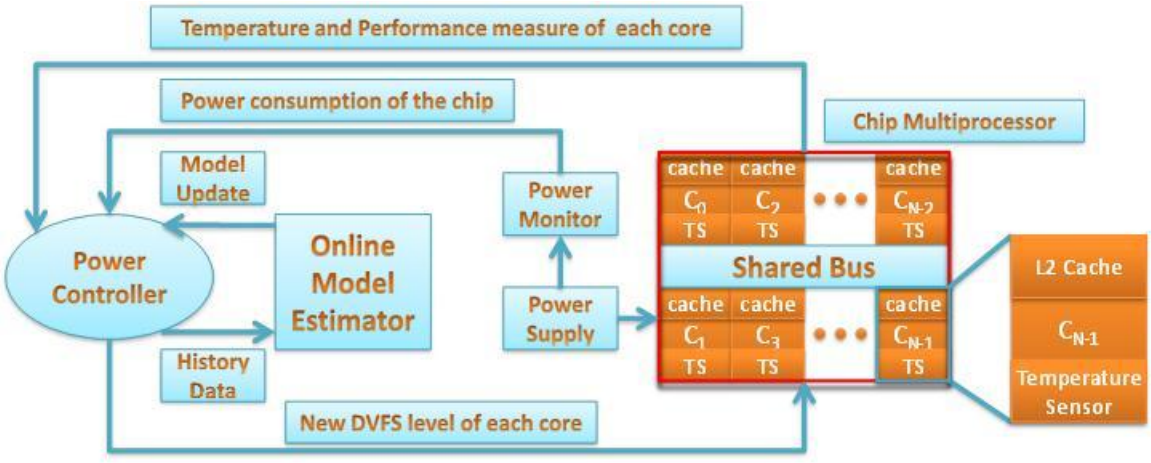


Figure 2.1 Feedback Control Loop for Temperature Aware CMP Power Control

$$p_i(k) = a_i f_i(k) + c_i \tag{2.2}$$

$$p_i(k+1) = p_i(k) + a_i \Delta f_i(k) \tag{2.3}$$

$$cp(k+1) = cp(k) + [a_1, \dots, a_n] \begin{bmatrix} \Delta f_1(k) \\ \dots \\ \Delta f_n(k) \end{bmatrix} \tag{2.4}$$

At control step *k*, the controller is going to decide the frequency settings (i.e. the manipulated variables) $\Delta \boldsymbol{f}(k), \Delta \boldsymbol{f}(k+1|k), \dots, \Delta \boldsymbol{f}(k+M-1|k)$, for the next *M* control steps such that the total chip power consumption (controlled variable) approaches the target power consumption following an exponential trajectory for the next *P* control steps while maximizing the system performance and satisfying the temperature constraints. *M* and *P* are called control horizon and prediction horizon respectively in MPC theory and they usually are equal. Based on these objectives, the cost function of the MIMO controller is designed as in (2.5). The first item in (2.5) is the difference between the chip power consumption and the reference trajectory, which represents the tracking error in MPC theory. And the second item represents the control penalty in MPC theory. The constraints for this optimization problem are the temperature constraint as well as the DVFS range limitation, which are all linear. Therefore at each control step, the controller just needs to solve a least-square optimization problem whose solution can easily be found using existing solvers.

$$\begin{aligned} V(k) = & \sum_{i=1}^{P} ||cp(k+i|k) - ref(k+i|k)||^2 \\ & + \sum_{i=1}^{Q} ||\Delta \boldsymbol{f}(k+i|k) + \boldsymbol{f}(k+i|k) - \boldsymbol{F}_{max}||^2 \end{aligned} \tag{2.5}$$

The authors of [61] implement the MPC controller on a real system with an Intel Xeon quad core processor and running Linux OS. The power measurements are obtained by a Digital Multimeter (DMM) which measures the current running through the power line to the processor. The temperature readings are obtained by the digital thermal sensor (DTS) residing in each core. The frequency scaling is enabled by the Intel Enhanced SpeedStep technology [1][2].

2.3 Predictive Temperature Aware Task Migration

Predictive DTM policies make control decisions based on projected future system dynamics. As long as the prediction is accurate, thermal emergency events can be avoided by taking appropriate actions in advance. Many predictive DTM policies have been proposed [20][17][63][10][19]. Their main difference lies in the adopted temperature prediction models and the DTM actions. In this section, we give a brief introduction of two predictive thermal management techniques [19][63]. Based on the adopted temperature prediction models, we refer them as “thermal aware task migration with RLS (Recursive Least Square) based temperature prediction” [63] and “thermal aware task migration with ARMA (Auto-Regression Moving Average) based temperature prediction” [19].

2.3.1 Thermal aware task migration with RLS temperature prediction

The rate of temperature change of an application depends on two factors: (1) the difference of its current temperature and its steady state temperature; and (2) the characteristics of the application itself. The first factor accounts for the long term thermal behavior while the second factor accounts for the short term thermal behavior. Based on this observation, the RLS based temperature prediction model consists of two parts: Application Based Thermal Model (ABTM) for the short term prediction and Core Based Thermal Model (CBTM) for the long term prediction.

The ABTM predicts future temperature by observing the recent thermal behavior of the application and incorporating this information into a recursive least square regression model. In its general form, the RLS model can be expressed in equation (2.6).

$$y = \theta_1 f_1(\boldsymbol{u}) + \theta_2 f_2(\boldsymbol{u}) + \cdots + \theta_n f_n(\boldsymbol{u}) \tag{2.6}$$

where y is the variable that is going to be predicted, $f_1, f_2, \cdots, f_n$ are fixed functions, and $\boldsymbol{u} = [u_1, u_2, \cdots, u_n]^T$ is the input vector of the model. In our case, y is the predicted future temperature, $\boldsymbol{u}$ are the n most recent temperature observations and $f_i(\boldsymbol{u}) = u_i$. Then equation (2.6) can be reduced to (2.7)

$$y = \theta_1 u_1 + \theta_2 u_2 + \cdots + \theta_n u_n \tag{2.7}$$

The $\boldsymbol{\theta} = [\theta_1, \theta_2, \cdots, \theta_n]$ is the unknown coefficient vector which is going to be estimated by the RLS model. To find out $\boldsymbol{\theta}$, we need a set of training data $\{(y_i, \boldsymbol{u_i}) | i = 1, \cdots, m\}$. Let $\boldsymbol{X} = [\boldsymbol{u_1}, \boldsymbol{u_2}, \cdots \boldsymbol{u_m}]^T$ and $\boldsymbol{y} = [y_1, y_2, \cdots, y_m]^T$, then the relation between the variables $\boldsymbol{X}, \boldsymbol{y}$ and $\boldsymbol{\theta}$ can be expressed in the following equation:

$$\boldsymbol{y} = \boldsymbol{X\theta} \tag{2.8}$$

And the coefficient vector can be solved by least square fitting using

$$\boldsymbol{\theta} = (\boldsymbol{X^T X})^{-1} (\boldsymbol{X^T y}) \tag{2.9}$$

When a new set of training data $(\boldsymbol{y_{m+1}}, \boldsymbol{u_{m+1}})$ arrives, it is not necessary to use (2.9) again to compute new $\boldsymbol{\theta}$ (denoted as $\boldsymbol{\theta_{m+1}}$), because the computation complexity is too high when the size of the matrix grows. Instead, we can use the following recursive equation to update

$\boldsymbol{\theta}$. Denote the $\boldsymbol{P_m} = (\boldsymbol{X^T X})$ when we have m training data. Then, given the new training data, the $\boldsymbol{P_m}$ can be updated using the following equations:

$$\boldsymbol{P_{m+1}} = \boldsymbol{P_m} - \frac{\boldsymbol{P_m u_{m+1} u_{m+1}^T P_m}}{\boldsymbol{1 + u_{m+1}^T P_m u_{m+1}}} \tag{2.10}$$

$$\boldsymbol{\theta_{m+1}} = \boldsymbol{\theta_m} + \boldsymbol{P_{m+1} u_{m+1}}(y_{m+1} - \boldsymbol{u_{m+1}^T \theta_m}) \tag{2.11}$$

In this way, the temperature prediction model is continuously updated as more training data is obtained during runtime. Please note that the dimension of the vectors and matrixes in (2.10) and (2.11) is always n, which is determined by the model order in (2.6). Therefore the computation complexity stays the same as new training data comes in.

The CBTM model accounts for the long term thermal behavior of the application. It neglects the short term power variation of the application and assumes that the application runs at constant power. CBTM again adopts the temperature change formula given by (2.12)

$$T(t) = T_{ss} - (T_{ss} - T_{init}) \times e^{-bt} \tag{2.12}$$

where T_{ss} is the steady state temperature, and b is a thermal constant. In this equation, T_{ss} are pre-computed for each application. b is also computed offline. It is calculated only once for all applications because it is determined by the thermal characteristics of the chip. Using (2.12), the processor temperature after time t can be predicted.

To predict the temperature at a future time t, the authors of [63] apply both ABTM and CBTM and obtain two predicted temperatures T_{ABTM} and T_{CBTM}. The final prediction is the weighted sum of these two as shown in equation (2.13).

$$T_{predict} = w_s T_{ABTM} + w_l T_{CBTM} \quad (2.13)$$

Based on the above temperature prediction model, the authors propose a thermal aware task migration policy called Predictive Dynamic Thermal Management (PDTM). When a task running on a processor is projected to exceed the temperature threshold, it will be moved to a processor that is predicted to be the coolest in the future.

The evaluation platform of this policy is an Intel Quad-core processor. Processor temperature can be read from the digital thermal sensor embedded in the cores. The applications running on the system are libquantum, perlbench, bzip2, hmmer from SPEC 2006 Benchmarks. The prediction model achieves very high accuracy and the temperature prediction error is only 1.6%. Compared to the existing thermal management scheme, the proposed task migration policy reduces the average temperature by 7% and peak temperature by 3^{o}C.

2.3.2 Thermal aware task migration with ARMA temperature prediction

The rationale behind the ARMA prediction model is that, when the workload is stationary, the temperature can be estimated accurately by regressing the past measurements. The formulation of the ARMA model, which is shown in equation (2.14), is similar to the RLS model because they are both linear regression based models.

$$y_t + \sum_{i=1}^{p} a_i y_{t-i} = e_t + \sum_{i=1}^{q} c_i e_{t-i} \quad (2.14)$$

y_i in (2.14) is the temperature at time t and the e_i is called prediction error or residual noise. In the equation, $\sum_{i=1}^{p} a_i y_{t-i}$ is called the auto-regression (AR) part and the $\sum_{i=1}^{q} c_i e_{t-i}$ is

called the moving average (MA) part. Similar to the RLS model, the coefficients a_i and c_i of the ARMA model are obtained through online training.

In contrast to the RLS model, the order of the AR model (i.e. p) and the order of the MA model (i.e. q) are not known at the beginning of training. It is obvious that a large p and q can achieve higher prediction accuracy, but it will also increase the complexity of the model thus the runtime overhead. To get a good balance between accuracy and complexity, the training process adopts a trial-and-error strategy. It starts from a small order, say $(p, q) = (1,0)$, and gradually increases the order until some accuracy measurement achieves a satisfactory extent. The accuracy measurement used here is called *Final Prediction Error* (FPE) which is defined in equation (2.15)

$$FPE = \frac{1 + n/N}{1 - n/N} \cdot V \tag{2.15}$$

In this equation, n is the model order which is equal to $p + q$. N is the length of the training time series and V is the variance of the model residuals. Because the FPE takes both training error V and model complexity n into consideration, it provides a good tradeoff between accuracy and complexity.

Another difference between ARMA model and RLS model is the way models get updated. In the RLS model, whenever a new training data arrives, the model will be updated. This means the model will be updated at regular time intervals, because the processor's temperature is sampled periodically. This could incur some unnecessary computation overhead if the workload is stationary. The ARMA model uses the Sequential Probability Ratio Test (SPRT) to determine if the workload is drifting from the previous stationary state to a new state which

means there is a workload pattern change. When the workload is in stationary state, the prediction error sequence is a white noise signal; the error follows a fixed probability distribution which has zero mean and a small variance (e.g. a Gaussian distribution). When the workload changes, the prediction error drifts from the previous distribution which can be detected by SPRT. Instead of simply testing the mean and standard deviation of the prediction error, SPRT performs statistical hypothesis tests on the mean and variance of the residuals.

In order to perform the hypothesis tests, a temperature prediction error history window is maintained online. When new temperature data arrive, the prediction error is computed and the history window moves forward, then the hypothesis test is performed by SPRT. Let the prediction error in the history window be denoted as $[e_1, e_{2,} \cdots, e_n]$, then the SPRT tests the following two hypotheses:

- H_1: e_i is drawn from a distribution function with mean M and variance σ^2.
- H_2: e_i is drawn from a distribution function with mean 0 and variance σ^2.

If SPRT decides H_1 is true, then it considers the prediction error drifts from the current distribution and the workload pattern has changed. Therefore a model updating is needed. On the other hand, if H_2 is true, then the workload pattern has not changed and is still stationary. Therefore, the ARMA model does not need to be updated. Please refer to [20] for more details in the SRPT test.

Compared to the RLS model, the ARMA model has several advantages. First, it reduces the runtime model updating overhead. Second, the model order in RLS is fixed and has to be set manually. While in ARMA model, the model order can be a variable. This flexibility enables the ARMA model to achieve better tradeoff between prediction accuracy and model complexity.

Third, as shown in the experiments in [20], the ARMA predictor achieves higher accuracy than the RLS predictor when they predict a longer time in the future. The authors also show that the prediction accuracy of the ARMA predictor is superior compared with two other predictors, i.e. exponential averaging predictor and history table based predictor.

Base on the ARMA predictor, the author proposed a Proactive Thermal Balancing (PTB) policy. Similar to the PDTM policy [63], this policy also moves the tasks from a processor which is predicted to be hotter to a processor which is predicted to be cooler. The difference is that PDTM moves the current running tasks while PTB moves the tasks in the waiting queue. The benefits of moving tasks in waiting queue is that when current running task finishes, the processor has a period of idle time to cool down. Therefore, hotspots can be avoided. As a mechanism of migrating a waiting task in the ready queues has already been implemented in the OS scheduler for load balancing purposes, this technique does not introduce additional implementation overhead.

Experimental results on an UltraSPARC T1 based processor model show that PTB reduces hot spot occurrences, spatial gradients, and thermal cycles by 60%, 80% and 75% respectively on average comparing to reactive thermal management. And it only incurs a performance cost of less than 2% with respect to the default scheduling policy for load balancing running on the system.

2.4 Chapter Summary

In this chapter, we survey four online dynamic thermal management techniques based on task scheduling, dynamic voltage and frequency scaling and task migration. Although the varieties of the application runtime characteristics and the uncertainties in user behaviors impose

significant challenges to online DTM, the techniques presented in this section are able to tackle thermal safety problems while still delivering high performance. While reactive techniques are able to respond to the changing of the system dynamics quickly, predictive techniques take actions in advance to avoid thermal emergencies.

Chapter 3 Dynamic Thermal Management for Multimedia Applications Using Machine Learning

As shown in Chapter 2, dynamic thermal management techniques have been widely studied and employed to control the temperature for different computing platforms, from servers [48], general purpose computers [20], to embedded systems [66]. These works consider applications of various characteristics, including web applications [20], standard benchmarks [17][31] and multimedia applications [55][64]. Among these, multimedia applications are expected to form the largest portion of workload in general purpose personal computers and portable computing devices like smart phones [55]. In spite of their popularity, the computation intensity of multimedia applications is likely to produce high temperature in these platforms [64]. A thermal safe solution is to run the applications at lower speed or reduce the computation by decreasing the *Quality of Service* (*QoS*). However these solutions impact the user satisfactory.

In this chapter, we consider the problem of dynamic thermal management for multimedia applications. We utilize the processor's dynamic voltage and frequency scaling (DVFS) ability to control the operating temperature under a threshold while maximizing the system performance, i.e. minimizing the CPU time of the multimedia applications. Our DTM technique does not require to pre-characterize the system for its thermal and power model neither does it need any

prior knowledge of the workload information. It relies on machine learning algorithms to find the best management policy during the runtime. Compared to the existing DTM techniques [17][55] it provides considerable performance improvements with marginal increase in the percentage of thermal hotspot.

We model the dynamic thermal management problem as a stochastic control process and adopt the reinforcement learning algorithm to find the optimum policy during runtime. As mentioned in Chapter 1, we consider the processor's DTM controller as a learning agent and model the rest of the system, e.g. the operating temperature, the hardware and the application status as the environment. After the learning agent, i.e. the DTM controller, takes an action (i.e. switching to a new operating frequency), it observes the environment and estimates the reward or penalty caused by this action. The agent learns from this experience and tries to improve its future action selections to maximize the reward (or minimize the penalty).

Most of the multimedia applications such as MPEG movie clips or MP3 files are naturally arranged into frames. The computation load of processing each frame has high temporal correlation. We exploit this property and perform the learning based DTM at a single frame granularity. The application status is characterized by its frame level computation load which can be obtained from the processor's performance counter.

The characteristics of the presented work in this chapter are summarized as the following:

- This is the first work that applies reinforcement learning algorithm to solve the problem of dynamic thermal management. The presented approach is truly adaptive. The learning agent does not require having any prior knowledge of the environment or the system power/thermal characteristics. It learns from the

experience and adjusts the policy online. Therefore, it works robustly in different computing systems.

- The presented learning algorithm explores the frame level temporal correlation in multimedia applications. The environment observation and policy adaptation are performed at single frame granularity.

- The presented learning model has very small run time overhead. It only incurs a few table lookups and some simple arithmetic operations.

- Instead of simply minimizing the thermal violation, our goal is to maximize the performance without increasing the thermal violation.

- The presented learning model is validated on a Dell Dimension 9200 desktop PC with Intel Core 2 processor. All the experimental results data reported in this chapter are gathered with the consideration of the real implementation and control overhead.

- Compared to running the application without dynamic thermal management and the existing DTM techniques that utilize the same event counter information, the learning based DTM provides better performance-thermal tradeoff.

The rest of this chapter is organized as follows: Section 3.1 reviews the related work. We discuss how to apply the reinforcement learning model in detail in Section 3.2. Experimental results are reported in Section 3.3. Finally, we conclude this chapter in Section 0.

3.1 Related Work

Dynamic thermal management (DTM) has been studied for different types of applications from general purpose industry standard benchmarks to multimedia applications. Reference [31] and [17] focus on thermal management techniques for SPEC benchmarks. The authors of [31] propose to dynamically adapt some micro-architecture parameters, such as instruction window size, issue width, and fetch gating level, to the application characteristics and hence control the processor temperature. The authors of [17] propose to utilize the processor's performance counter readings to detect the phase changes of application at run time and adjust the operating frequency accordingly to avoid thermal violations. Both these works perform the thermal management at a regular time interval. Although this is effective for standard benchmarks, as we will show in Section 3.2, it might not be the same for multimedia applications.

There have also been a number of studies of thermal management for multimedia applications. Reference [64] proposes to profile the number of cycles to decode each frame. With this information and a temperature prediction model, an operating frequency is selected for a group of frames to guarantee the QoS while minimizing the temperature. Our work is different from [64] as we try to maximize performance while stratifying the thermal constraint. The thermal management scheme in [55] is based on the observation that the same type of frames has similar average IPC and power consumption, so the same configuration is applied to these frames.

One common drawback for the aforementioned works is that they rely on certain system models or profiled information. For example, [31] utilizes a neural network model to predict future temperature for a set of architecture parameters and application characteristics while [17] uses a linear regression model to predict future temperature. Even though those prediction models are carefully characterized, they suffer from lack of adaptability. Once the model has

been trained or obtained, it cannot be modified. Any change in the system or environment (e.g. room temperature variation, chip aging) will invalidate the model and thus hit the performance of the thermal management scheme.

Model free online learning techniques should be a solution to this problem. Previous work [58] has successfully applied the online learning model to dynamic power management (DPM) problems, but few works have applied the learning techniques to the DTM problem.

3.2 Reinforcement Learning For Thermal Management

In this section, we will first give the formulation of Q-learning in its standard form in Section 3.2.1 and then discuss how to apply this technique to the thermal management problem in Sections 3.2.2 ~ 3.2.5.

3.2.1 The Reinforcement Learning Model

Reinforcement learning [56] is an unsupervised machine intelligence approach that has been applied in many different areas. The general learning model consists of

- An agent, with a finite action set $\boldsymbol{A}$.
- The environment that has a finite state space $\boldsymbol{S}$. The actions of the agent will affect the future states of the environment, which in turn affects the options and opportunities available to the agent at later times.
- A policy $\boldsymbol{\pi}$ that defines the behavior of the learning agent at any given time. It is a mapping from the set of environment states to the set of actions, i.e. π: $\boldsymbol{S} \rightarrow \boldsymbol{A}$.

- A reward function $\boldsymbol{r}$: $\boldsymbol{S} \times \boldsymbol{A} \rightarrow \boldsymbol{R}$ which maps each state-action pair to a single real number, ($\boldsymbol{R}$ is the real number set). A *reward* indicates the intrinsic desirability of taking an action at one particular state.

The process of reinforcement learning is divided into multiple decision steps. We refer to each decision step as *decision epoch*. At each *decision epoch*, the agent takes an action according to current environment state. It then observes the environment for the reward/penalty caused by this action. The goal of the agent is to maximize the average long-term reward by trial-and-error interaction with a dynamic environment. It is achieved by learning the policy π, i.e. a mapping between the states and the actions. The agent might not select the optimum action at beginning, but its performance can be improved over time.

The *Q*-learning algorithm is one of the most popular reinforcement learning algorithms. In *Q*-learning, the agent keeps a value function $Q^{\pi}(s,a), s \in \boldsymbol{S}, a \in \boldsymbol{A}$, for each state-action pair and stores it in a *Q*-table. The value function represents the expected long-term reward if the system starts from state *s*, taking action *a*, and thereafter following policy π. Based on this value function, the agent decides which action should be taken in current state to achieve the maximum long-term rewards. An optimum policy $\boldsymbol{\pi}^*$ is a policy which achieves the maximum value function denoted as Q^*, i.e. $Q^* = Q^{\pi *}(s,a) = \max_{\pi} Q^{\pi}(s,a)$.

The core of the *Q*-learning algorithm is to iteratively update the value function $Q^{\pi}(s,a)$ as following [56]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times \left[r_{t+1} + \gamma \times \max_{a} Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

In the above equation, α is the learning rate which determines how fast the Q value will adapt. The discounted factor γ is between 0 and 1.

3.2.2 Frame Based Decision Epoch

How frequent an agent observes its environment, updates the policy and issues DTM command is determined by the length of the decision epoch. An appropriate decision epoch can help the system to learn and control effectively. Within a decision epoch, the system should have consistent behavior. Across multiple decision epochs, the system behavior should exhibits repeated patterns. Here we define a *workload phase* to be an execution interval during which the application has near identical power, temperature and performance characteristics [17]. It is obvious that the workload phase of an application determines the decision epoch of its DTM agent.

For a multimedia application, the decision epoch can be divided in two ways: frame based decision epoch and equal time step decision epoch. An equal time step decision epoch at the granularity of 100 *ms* has been used for the DTM of SPEC CPU2006 benchmarks [17]. This is because the workload phase change of the applications in SPEC CPU2006 benchmarks can be detected at this granularity. (Those applications will stay in the same workload phase for several to tens of seconds before moving to next phase [17].) However, the same equal time step decision epoch is not suitable for multimedia applications. Figure 3.1 shows a segment of trace of retired instructions for the MPEG4 decoder from the MediaBench [6]. The upper part and lower part of the figure show at time interval of every 10 ms and every 5 ms respectively. The number of retired instructions has been reported as one of the architectural events that contribute most for the temperature change [17]. From this figure, we can see that instruction retired number is constantly vibrating and does not show obvious phase change. This is because equal

time epoch smoothes out the inner workload phase change of the application. On the other hand, Figure 3.2 demonstrates the trace of the retired instructions every frame for a total of about 200 frames. As shown in this figure, phase change can be clearly observed with a regular pattern. Therefore in this work we choose frame based learning epoch.

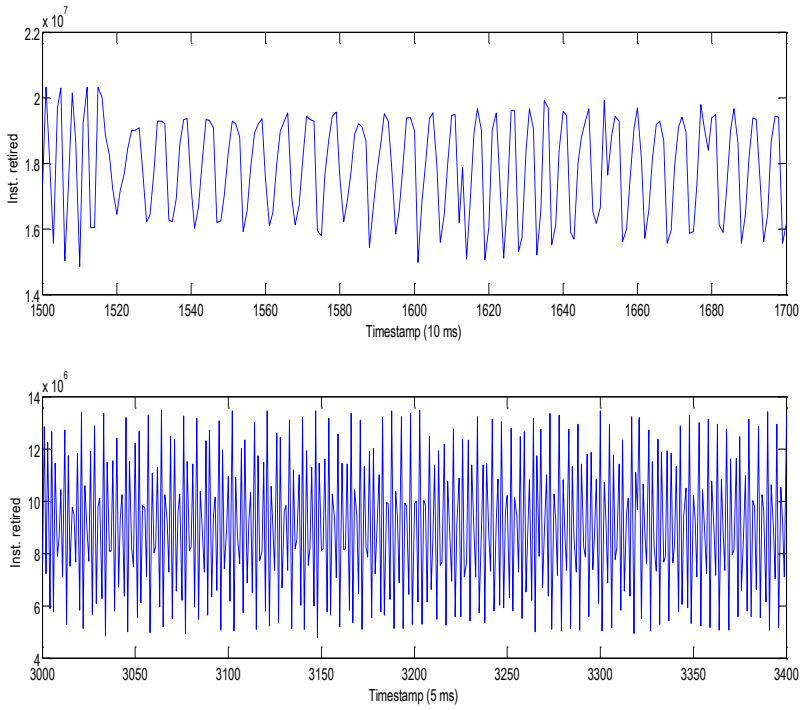


Figure 3.1 Variation of retired instructions for every 10 and 5 ms for MPEG4 decoder

3.2.3 Interactions Between Agent and Environment

Figure 3.3 presents the system model of our learning based DTM agent. The actions that the agent takes are the available frequency levels of the processor. At each decision epoch, the agent observes the current state of the environment and chooses the frequency level for the next frame according to the *Q*-values in the *Q*-table. The Intel's speedstep technology [2] or AMD's

PowerNow technology [1] can be used to dynamically switch the voltage and frequency of a processor.

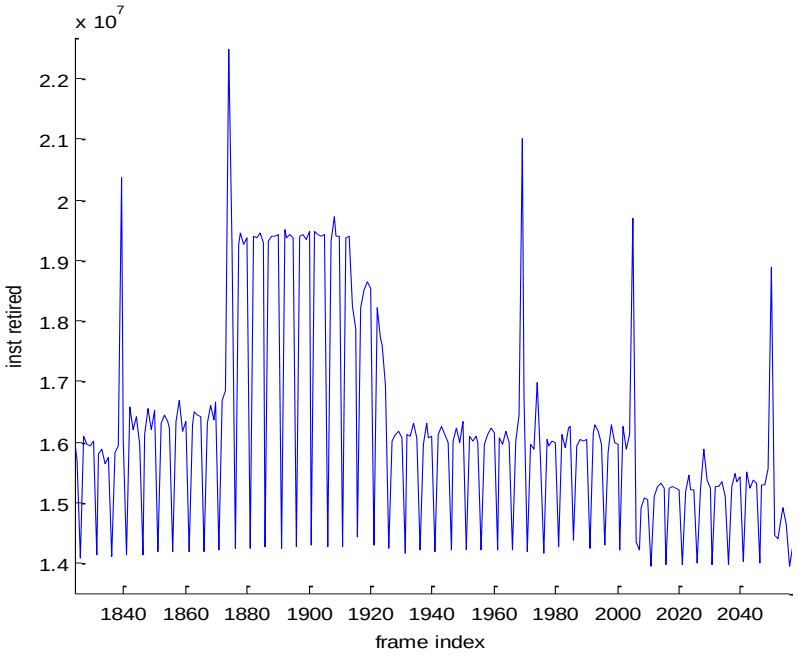


Figure 3.2 Variation of retired instructions at frame granularity

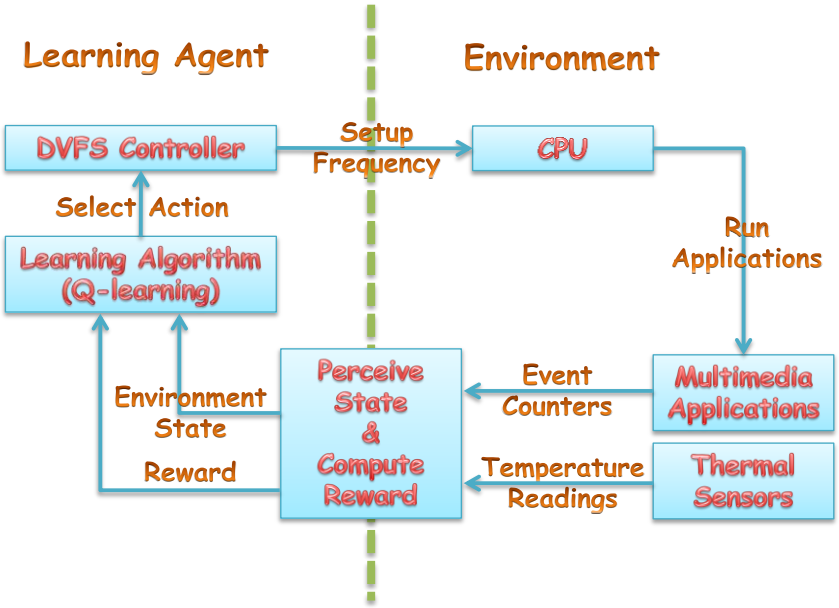


Figure 3.3 System model

Everything outside the agent is considered as environment, including the processor, the applications and the thermal monitoring system (thermal sensors). The environment and the agent are closely coupled. For instance, the power consumption of the processor is linearly dependent on the operating frequency, which in turn determines the processor's temperature. As another example, because the agent only changes the clock frequency of the processor not the clock frequency of the memory subsystem, its actions will affect the instruction per cycle (IPC) of the system. On the other hand, the environment status limits the agent's actions in some cases by changing the reward/penalty value. For example, the agent is not allowed to run at full speed when the processor is approaching the temperature threshold.

3.2.4 Classification of Environment States

The environment can be characterized by many features, from the temperature and the power consumption of the processor, to the cache miss rate and IPC. The learning agent works under a discrete state space. How to map this huge and sometimes continuous feature space into a finite set of discrete state space has direct impact to the effectiveness of the learning algorithm. Here, two problems are involved.1) Which features should be selected to represent the environment; 2) How to discretize the selected feature space into the state space.

The rules of thumb of selecting features to represent the environment are: first, those features should closely relate to our problem, i.e. performance optimization with thermal constraint; second, they could be observed easily.

It is obvious that the processor's temperature should be one of the features to represent the environment state as it is directly related to our optimization problem. This information can

be obtained by reading the temperature sensors that are equipped on most state-of-the-art processors.

We may also want to use the processor power consumption as one of the features for environment state, because it directly contributes to the temperature change of the processor. Unfortunately, this information is not easy to obtain. Therefore, we decided to use the readings of performance counters as a proxy for the power consumption as they reflect the processor's switching activities. Besides power consumption, the performance counter readings, such as the execution cycles, instruction retired rate and cache miss rate, etc., also reflect the performance of the application programs, which is what we want to optimize.

Our test platform, the Intel's Core 2 Duo processor has 5 performance counters which could record 130 architectural events [7]. Among them, we select those events that contribute most to the temperature change and are most relevant to the system performance. Reference [17] utilizes the principle component analysis to find the contribution of each architectural event to the temperature change and suggests that 3 of them, i.e. the "instruction retired" event, the "floating point instructions executed" event and the "conditional instructions executed" event, play the most important roles in temperature change. However, our analysis shows that, for a typical multimedia application such as an MPEG-4 decoder, the variation of floating point instructions executed among different frames is very small. We also found that the number of conditional instructions executed has high correlation with the number of retired instructions and thus does not provide much additional information.

We analyzed the correlations between different events and retired instructions that were recorded during an MPEG decoding process. TABLE 3.1 Correlation between different events

and retired instructions gives the selected results. The results indicate that the "last level cache miss" event has the least correlation with the "instruction retired" event and hence provides the most additional information about the system. Therefore we use the "last level cache miss" event together with the "instruction retired" event to represent the environment state. The former represents the memory activities while the latter specifies the computation activities. Combined together they provide a complete picture of the system. This choice also agrees with what is suggested in reference [32].

TABLE 3.1 Correlation between different events and retired instructions

UOPS RETIRED	BR_INST RETIRED	BR_CND EXEC	FP_INST EXEC	BUS_MEM TRANS	LAST_LEVEL $_MISS
0.997	0.977	0.964	0.763	0.762	0.669

Based on the above analysis, we will use the feature set (T, $\boldsymbol{P}$) to characterize the environment, where T is the reading from the thermal sensors and $\boldsymbol{P}$ is a vector of selected event counter readings. In the next, we will discuss how to map the feature space into a finite state space to apply the reinforcement learning model.

Note that T can be any real number within the working range, usually from $0^oC \sim 100^oC$. Because it is a continuous variable, we have to discretize it to get a finite set of states. We divide the temperature working range into a set of disjoint intervals, i.e. $(0,\ T_0]$, $(T_0,\ T_1]$, $(T_1, T_2]$, … , $(T_{N-1}, T_N = T_{threshold}]$, (T_N, ∞), each interval $(T_{i-1},\ T_i]$ corresponds to a state T_i. Note that the (N+1)th interval covers all temperatures beyond the threshold. Although the temperature level $T_0, \ldots T_{N-1}$ can be set arbitrarily, we would divide the region near the threshold temperature in finer granularity while leave the other region in coarse granularity. In this way when the temperature is approaching the threshold, the agent might take better control at finer resolution.

In order to classify the space of the event counter readings (i.e. ***P***) we use the k-means clustering method as in [17]. We took 5 representative video clips and collect the retired instructions number and cache miss number for each frame, and classify them by the standard k-means algorithm [57]. To find the optimum number of states (i.e. number of clusters), we start from a small number and gradually increase it until the classification error is less than 5%, which is defined as the ratio between square sum of all points' within cluster distance and the square sum of their distance to the origin. Based on k-means clustering, we divide the event counter values into K states $\{P_0, \cdots, P_{K-1}\}$. Together with the N temperature states, the size of the resulting learning space is $|N| \times |K|$. In our implementation, we set N=11 and K=10.

3.2.5 Design of the Reward Function

The state of the ith decision epoch is denoted as (T^i, p^i), where T^i give the temperature at the beginning of the ith epoch and p^i gives the number of cache misses and the number of instruction retired during the ith epoch. The action taken by the agent during the ith epoch is denoted as a^i. At the end of the ith epoch (or the beginning of the $(i+1)$th epoch, the agent calculates the reward caused by the action and update the Q-function of state action pair (T^i, p^i, a^i). The reward function is defined as the following:

$$r\left((T^i, p^i), a^i\right) = \begin{cases} -PT, & if\ T^{i+1} = N+1 \\ A \cdot Inst\left(p^{i-1}\right) \cdot Freq\left(a^i\right) & \\ +B \cdot T^{i+1}, & if\ T^{i+1} \neq N+1 \end{cases} \tag{3.1}$$

where T^{i+1} is the temperature state at the end of the ith epoch (or the beginning of the (i+1)th epoch), $Inst(p^i)$ is the number of retired instructions, $Freq(a^i)$ is the processor frequency selected by action a^i, and state (N+1) is the temperature state that covers all temperatures beyond

the threshold. In this function, *P*, *A* and *B* are constants. The upper part of the reward function is the thermal violation penalty, which is a negative number. If at the end of the *i*th epoch, we reach a thermally unsafe state (i.e. $T^{i+1} = N+1$), then a negative reward will be received. A negative reward will decrease the *Q*-value of state action pair (T^i, p^i, a^i) so that this action will be avoided at this state in the future.

The lower part of the reward function is used when the system is thermally safe at the end of the *i*th epoch. In this scenario, we would like to increase the performance of the system. The first part of the reward function is the performance. We use the product of the number of retired instructions and the processor frequency to represent the performance reward. This encourages the agent to select high frequency for frames with high computation demand. The intuition behind this is that use high frequency for complex frames can reduce more execution time than using it for simple frames. The second part of the reward function represents the thermal award. The parameters A and B provide tradeoff between temperature and performance.

Generally, the convergence of the learning process depends on the recurrent visits of all possible state-action pair. We have several techniques to improve the convergence speed of the learning process. First, our learning policy encourages using higher frequency in each state as long as it is thermal safe. Therefore, the action space is reduced. So the number of recurrent visits on each action is increased. Second, the virtually visiting technique and variable learning rate technique proposed in reference [54] can be applied to further increasing the convergence speed. For example, if the state-action pair is $((T_i, p_i), a_i)$ and the next state is (T_{i+1}, p_{i+1}). Apparently, p_{i+1} is not dependent on the action. If T_{i+1} is in the thermal violation region, then we could update all state-action pair $((T_i, p_i), a_j)$, such that the frequency of a_j is bigger than that a_i.

3.3 Experimental Results

3.3.1 Experiment Setups

We carried out our experiments on a Dell Dimension 9200 desktop with Intel Core 2 Duo E6400 processor which has 2MB L2 cache and 1333 MHz FSB. The processor supports 8 frequency levels: 267 MHz, 533 MHz, 800 MHz, 1.07 GHz, 1.33 GHz, 1.63 GHz, 1.87 GHz and 2.13 GHz. The operating frequency of each core in the dual core processor can be adjusted separately. The operating system is Fedora 11 with Linux kernel 2.6.29.

To monitor the temperature change of the processor, we utilize the thermal sensors available in each core [61]. The *coretemp* driver in the Linux kernel generates an interface under */sys/devices/ platform/coretemp.[X]* directory (X is the index of each core), and the current temperature will be reported when the file *temp1_input* is read each time. The default driver only updates temperature readings once every 1 second. This granularity is too coarse for our frame based management scheme, because the decoding time for a frame is about tens of milliseconds. We modified the driver so that it could update its readings every 10 ms.

We utilize Intel Enhanced SpeedStep [2] technology to adjust the frequency level. Similar to temperature readings, the Linux kernel provides the *cpufreq* driver for users to read and modify the operating frequency. Processors equipped with DVFS ability will have an interface under */sys/devices/system/cpu/cpu[X]/cpufreq/* directory (X is the index of each core). It has been reported in [61] that the overhead for each frequency adjustment is about 20 us, therefore the overhead for DVFS at every frame is under 2%.

To collect the performance counter readings, we utilize the pfmon 3.9 hardware monitoring tool [7]. Two event counters are monitored in our program, i.e. the instruction retired event and cache miss event. The monitoring is trigger at the end of each frame.

We choose the MPEG-4 decoder from the MediaBench benchmark [6] as our application. Please note that the presented method can be readily applied to other multimedia applications, such as MPEG-2 decoder, H.264 decoder etc, because they have similar characteristics. We apply MPEG-4 decoder on 5 video clips extracted from recent movies of different genres, e.g. drama, action, animation.

We compare our reinforcement learning based DVFS thermal management policy with the Phase-Aware dynamic thermal management policy proposed in [17]. In Phase-Aware DTM, performance counter values are collected every 100 ms. Then the readings are classified into different phases. Based on a linear temperature prediction model, the max frequency which is guaranteed to be thermal safe under current phase will be selected. It is important to point out the effectiveness of the phase aware DTM heavily relies on the accuracy of the temperature prediction model. We also compare our policy with two scenarios that run the entire application at 1.63 GHz and 1.87 GHz clock frequencies without any dynamic thermal management.

Please note that although we use MPEG-4 decoder from MediaBench and the Dell 9200 desktop as our platform, the proposed reinforcement learning model is not only limited to these specific application and platform. As long as we choose the proper decision epoch, the appropriate features to characterize the environment and an effective reward function, the proposed learning model could readily be applied to general applications under other architectures. For example, for a general benchmark, there is no frame based decision epoch. In

this case, an equal time interval decision epoch would be appropriate. On our current platform, the decoding process is completely implemented in software and runs on the CPU. Therefore, the control action is to change the frequency of the CPU. However, if the multimedia application runs on a IP-rich System-on-Chip, while the computation is mainly offloaded to the IP core, the control action could be changing the frequency of the audio or video codec.

3.3.2 Results Analysis

In the first set of experiments, we compare the run time and thermal violations among our reinforcement learning based policy, Phase-Aware policy, and two single operating frequencies and the results are shown in Figure 3.4 and TABLE 3.2. The thermal violation is defined as the percentage of time that the processor temperature is above the given threshold. We use 1.63 GHz as the base line frequency and set the peak temperature under this frequency as the thermal threshold. The reason that we use a floating temperature threshold instead of a fixed temperature threshold is to cancel the impact of the ambient temperature, which cannot be controlled by us. We do not distinguish the violation temperatures because we found in our experiments, all violation temperatures are within 5^{o}C. Using the next available higher level frequency could reduce the run time significantly; however, without any thermal management, it also incurs large thermal violations. The average thermal violation for 1.87 GHz is 36.65%. On the other hand, learning based policy provides large performance improvement with very small thermal violation. For example, our Learning policy improves the run time by 22.15% and 7.53% over the 1.63 GHz policy and the Phase-Aware policy, while only incurs 2.38% thermal violation. And compared to the 1.87 GHz clock frequency, the Learning policy reduces thermal violation significantly while maintaining the similar run time. The reason that the Learning policy has marginal higher thermal violation than the 1.63 GHz and the Phase-Aware policy is because,

unlike the Phase-Aware policy, it does not employ a temperature prediction model and has to try frequency settings at different states. Therefore, the Learning policy will make some mistakes "on purpose" in order to learn the optimum policy.

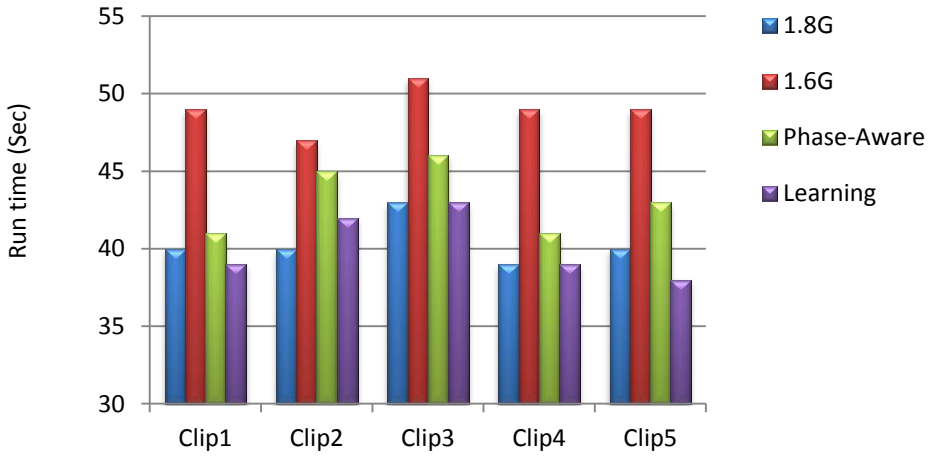


Figure 3.4 Comparison of run time for different policies

TABLE 3.2 Comparison of thermal violations in percentage of time

Policy	Clip1	Clip2	Clip3	Clip4	Clip5
1.6G	0	0	0	0	0
Phase-Aware	0.49	0.4	0.65	0.25	0
Learning	1.78	3.31	2.83	1.26	2.7
1.8G	33.66	60	21.95	42.65	24.98

We observed in our experiments that Learning based policy is more aggressive than the Phase-Aware policy. This is illustrated in Figure 3.5 Temperature and frequency comparison between Learning and Phase-Aware, which shows the temperature variation (upper part) and operating frequency (lower part) for an interval of 100 frames. The green circle line shows the data for the Phase-Aware policy while the red dashed line shows the data for the learning policy. We also plot the number of instruction retired for each frame in the lower figure (the blue line). As shown in the figure, both control policies run at the thermal threshold 48^{o}C for most of the

time, while the Learning policy incurs minor thermal violation. The Phase-Aware policy fix the frequency at 1.87 GHz. while Learning based algorithm is able to perceive the state change at the frame granularity and learns a control policy that alternates the clock frequency between 2.13 GHz and 1.87 GHz.

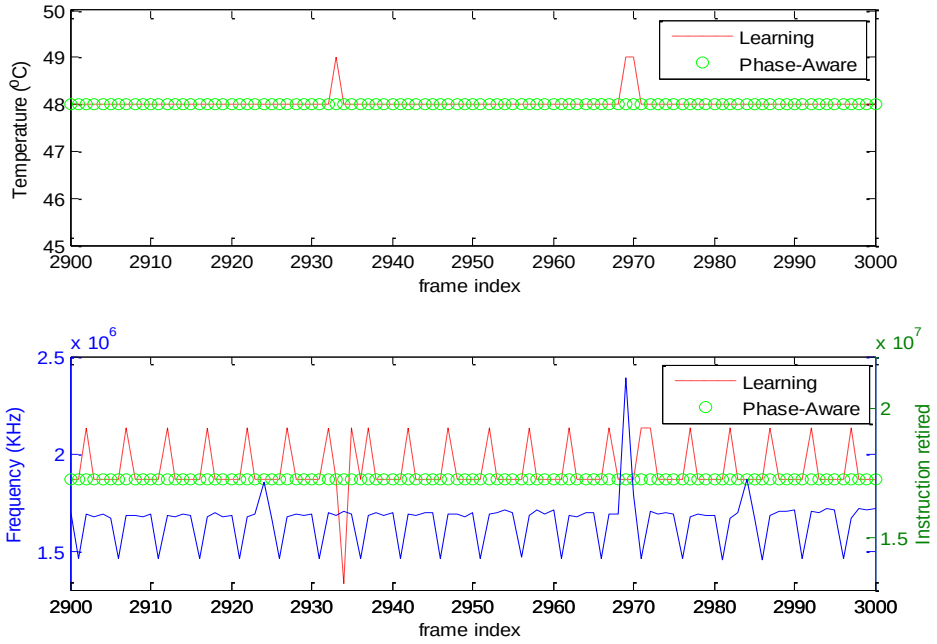


Figure 3.5 Temperature and frequency comparison between Learning and Phase-Aware

Figure 3.6 shows the percentage of time that Learning and Phase-Aware policies run at each frequency. As shown in the figure, learning policy was able to run at the highest frequency for more than half the time, while the Phase-Aware policy is more conservative and runs at the second highest frequency for most of the time.

Our Learning policy is very flexible. We can achieve different performance-thermal violation trade-offs by changing the parameters in the reward function. In the second set of experiments, we vary the thermal violation penalty *PT* in the reward function from 500, 1000 ~

10000 by a step of 1000 and obtained a set of trade-off points which are shown in Figure 3.7. The value of *PT* is also shown in the figure. When the thermal violation penalty is small, the learned control policy tends to be more aggressive. On the other hand, when the penalty is large, the learned control policy tends to be more conservative. This trade-off is a unique property of our Learning policy and could not be achieved by the Phase-Aware policy. This property could be useful when the reliability is considered as a resource that can be used to trade for performance improvement.

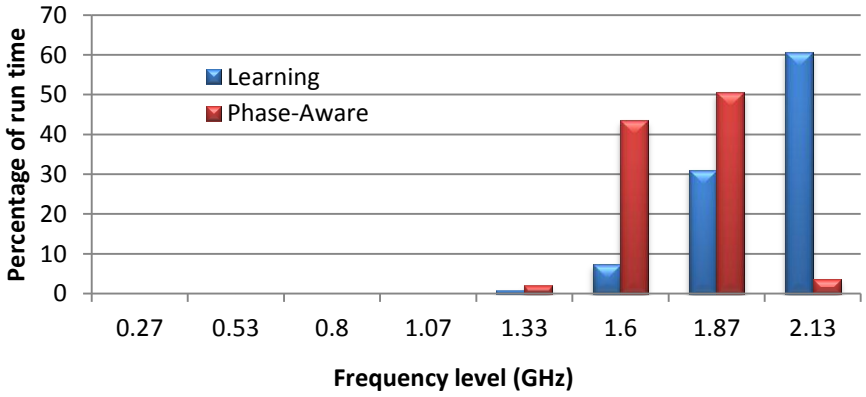


Figure 3.6 Percentage of time of each frequency for Learning and Phase-Aware policy

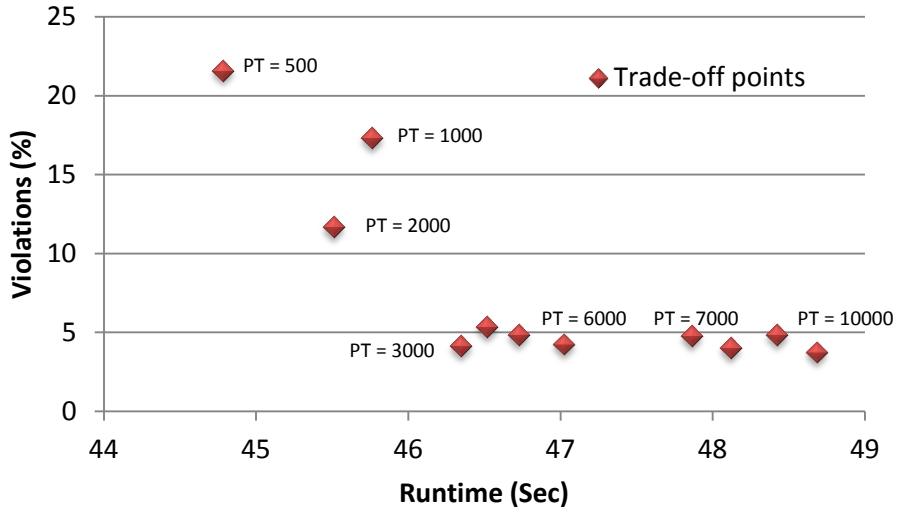


Figure 3.7 Trade-off between run time and thermal violation

In the last set of experiments, we tested the impacts of different environment state representations to our learning policy. We tested three kinds of state representations: retired instructions with cache misses, retired instructions only and cache misses only. We applied the same Q-learning algorithm using these state representations and compared the run time and thermal violations. TABLE 3.3 shows that combining these two variables together could result in optimum performance. Compared to systems using only retired instructions or cache misses, in average, it reduces the run time by 3.82% and 5.16% respectively and also has 0.974% and 1.108% lower thermal violations respectively. The results indicate that the learning agent receives more information about the environment when monitoring these two variables, and hence makes better control decisions.

TABLE 3.3 Comparison of different state representation

Clip #	State Representation	Inst + Cache	Inst only	Impr. (%)	Cache only	Impr. (%)
Clip 1	Violations (%)	3.80	4.65	0.85	5.58	1.78
	Run time (Sec)	40.70	42.61	4.71	43.53	6.96
Clip 2	Violations (%)	4.90	5.16	0.26	4.94	0.04

	Run time (Sec)	41.02	40.90	-0.28	42.81	4.37
Clip 3	Violations (%)	5.24	6.30	1.06	5.47	0.23
	Run time (Sec)	42.81	44.79	4.63	47.49	10.95
Clip 4	Violations (%)	3.66	5.90	2.24	6.25	2.59
	Run time (Sec)	45.32	47.40	4.60	45.20	-0.27
Clip 5	Violations (%)	2.53	2.99	0.46	3.43	0.90
	Run time (Sec)	37.35	39.38	5.45	38.77	3.79

To compare our algorithm with other learning algorithm, we refer to the reference [27]. It proposed a reinforcement learning based algorithm similar to us with multiple optimization objectives, i.e. temperature, power and performance. And it compare with an expert-based learning algorithm proposed in [23]. Their results show that when the temperature constraint is tight, reinforcement learning based algorithm could finish the application faster than the expert-based learning algorithm.

3.4 Chapter Summary

In this chapter, we presented reinforcement learning based dynamic thermal management method for multimedia applications. The agent learns the workload pattern of the application based on the performance counter readings, and adjusts the processor's operating frequency at the beginning of each frame to optimize the performance while ensuring thermal safety. We implemented our learning based DTM policy on a personal computer and tested it using real application. Our experimental results show big performance improvement with only marginal thermal violations compare to a thermal management policy that also based on workload phase detection.

Chapter 4 Dynamic Thermal Management for Many-core System

Most of these existing techniques [18][20][31][42][45][46][52][61] presented before or in the literature 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. Secondly, 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 [25]. 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 chapter we present 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 presented framework achieves much better scalability than the centralized approach. We refer to the presented technique as *distributed thermal balancing migration* (DTB-M) as it aims at balancing the workload and temperature of the processors simultaneously.

A steady state temperature based migration (*SSTM*) scheme as well as a temperature prediction based migration (*TPM*) scheme are presented in this chapter. 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 presented in this chapter. 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 [20][63], 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. Secondly, 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 ([20] and [63]) 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 ([20][63]), the neural network based peak temperature predictor works more robustly especially during the time when the workload changes, which usually happens after task migration.

Comparing to the existing proactive thermal-aware task migration, the presented 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 [20]. Furthermore, the DTB-M also has much lower migration overhead due to its distributed nature.

Comparing to the work in [26], this work provides the following two major extensions.

The first major extension of the chapter 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 [26]. (3) We compared our neural network model with an improved *auto-regression moving average* (*ARMA*) prediction model proposed in [20]. 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 chapter is the enriched experimental results section. We investigated the impact of different prediction models on the efficiency of the presented 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 [20].

The rest of the chapter is organized as follows: Section 4.1 reviews the previous work. Section 4.2 gives the semantics of the underlying many-core system and the application model. We give an overview of our thermal management policy in Section 4.3, while the detailed prediction model and migration schemes are presented in Section 4.4 and 4.5 respectively. Experimental results are reported in Section 4.6. Finally, we conclude this chapter in Section 4.7.

4.1 Related Work

In a many-core system, the heat dissipation capability differs from processor to processor. In [37] an algorithm is proposed to map and schedule tasks based on the thermal conductivity of different processors. In [52][42], the authors 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. The author in [42] 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 [50], 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 [51]. The authors 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 [20] and [63]. 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. Secondly, 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 [20] and [63], 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.

4.2 System Infrastructure

A tile-based *network-on-chip* (*NoC*) architecture [22] 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 chapter. 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 presented 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 [29][41]. 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 [24].

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 (Network-on-Chip) systems [43][47]. Examples of such system are Intel's single-chip cloud computing (SCC) platform [28] and RAMP (Research Accelater for Multiple Processors) [62].

The presented 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 presented 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 ([13] and [30]) 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.

4.3 Distributed Thermal Management Policy

In this section we present the details of the distributed proactive thermal balancing migration (DTB-M) policy. TABLE 4.1 summarizes the notations that will be used in this chapter.

TABLE 4.1 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

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 [15] that more than 95% accuracy can be achieved in power estimation using information provided by performance counters that are available in many modern processors. In the rest of the chapter, 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.

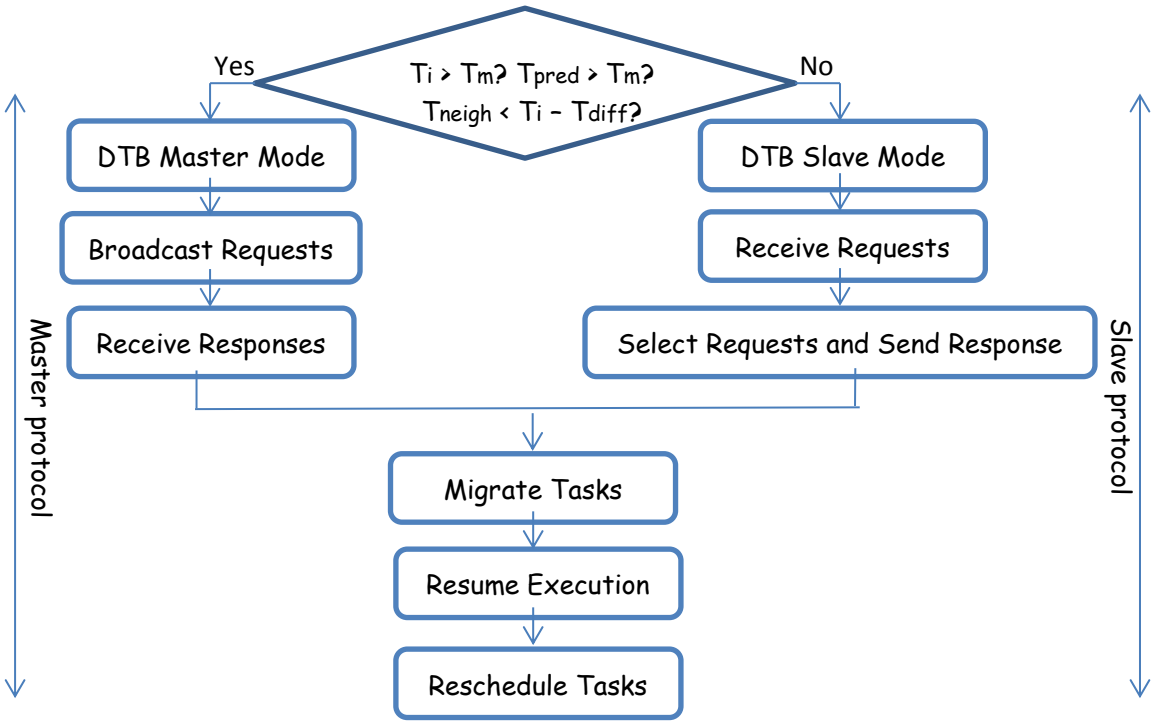


Figure 4.1 Master-Slave execution protocol

The DTB-M policy basically can be divided into 3 phases: temperature checking and prediction, information exchange and task migration. Figure 4.1 shows the flowchart of the DTB-M execution in the *i*th 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. Please note the master is equivalent to the sender and the slave is equivalent to the receiver in other multi-agent context. 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-δ 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.

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 Figure 4.2. 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.

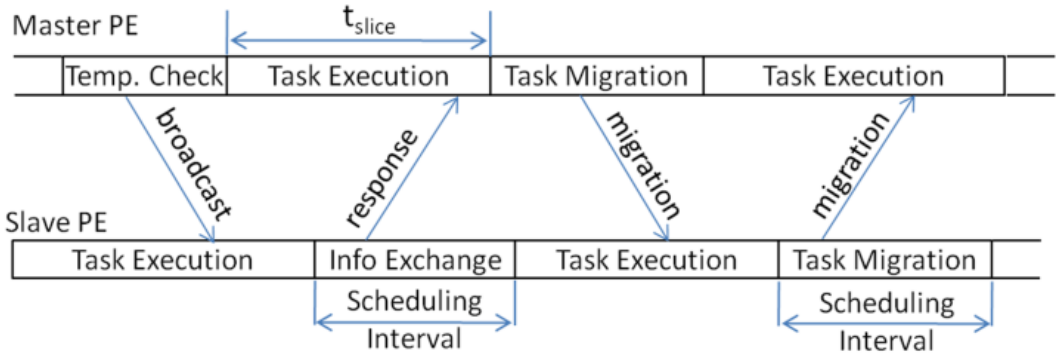


Figure 4.2 Master-slave communication

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 $\left||LT_i| - |LT_j|\right| > nt_{diff}, j \ni N_i$. The detailed workload balancing policy is presented in section 4.5.4. 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 Steady State Temperature based Migration policy (SSTM). It distributes tasks to cores based on their different heat dissipation abilities. The second technique is a Temperature Prediction based Migration policy (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.

4.4 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. Subsection 4.4.1 presents our neural network prediction model. Subsection 4.4.2 extends the ARMA prediction model proposed in [20]. And Subsection 4.4.3 compares the performance of the two prediction models.

4.4.1 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, Figure 4.3 shows a 12 seconds 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 50ms. 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^{o}C and it changes rapidly while the peak temperature changes much slower and the variation is less than 2^{o}C. Similar observation has been reported in reference [16]. Our second observation is that the peak temperature strongly depends on the task combinations running on the PE. As shown in Figure 4.3, 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 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.

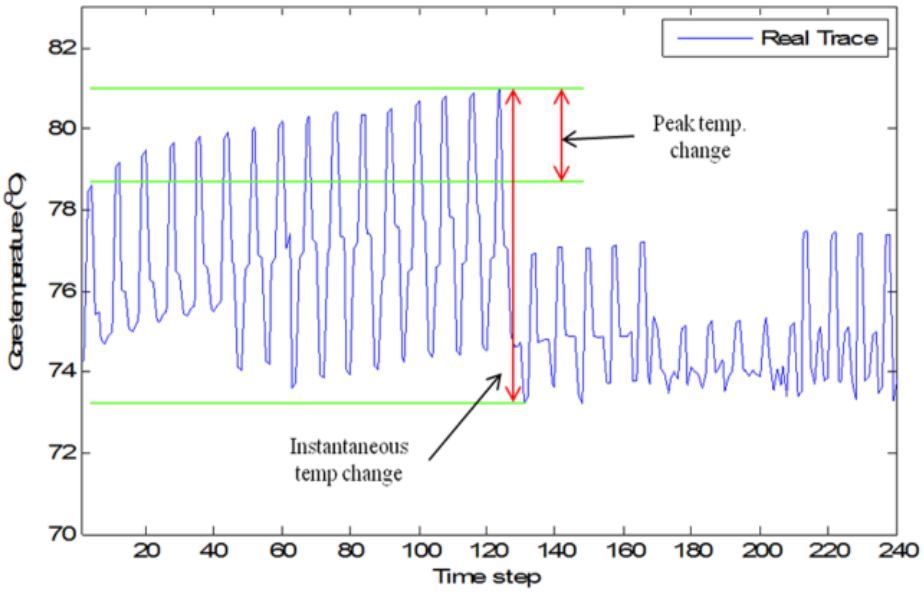


Figure 4.3 Example of instantaneous and peak temperature change

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 [9]. 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 Figure 4.4. 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 equation (4.1), 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.

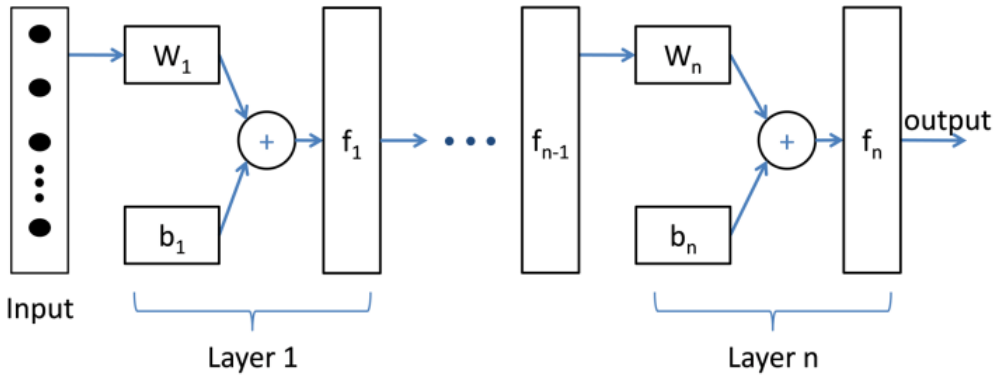


Figure 4.4 Neural network structure

$$a = f_n(\boldsymbol{W}_n f_{n-1}(\cdots f_1(\boldsymbol{W}_1 \boldsymbol{p} + \boldsymbol{b}_1)) + \boldsymbol{b}_n) \tag{4.1}$$

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

In this chapter, a two-layer neural network as shown in Figure 4.5 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.

$$tansig(x) = \frac{2}{1 + \exp(-2x)} - 1 \tag{4.2}$$

$$purelin(x) = x \tag{4.3}$$

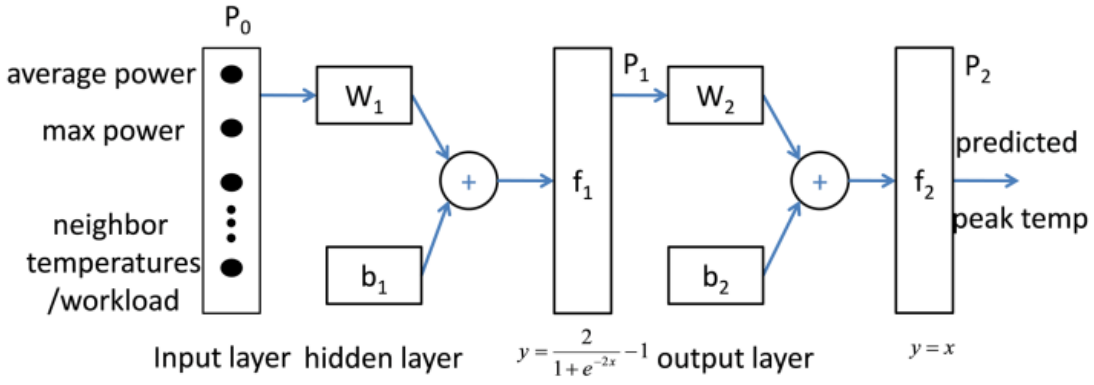


Figure 4.5 Neural network predictor architecture

A set of features relevant to the peak temperature prediction are selected as the inputs to the neural network. They can be divided into 2 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 *i*th 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 3 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 [31] 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 chapter 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 4.6. 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.

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 4.2 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.

TABLE 4.2 Prediction Accuracy vs. 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

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 4 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 4.6.

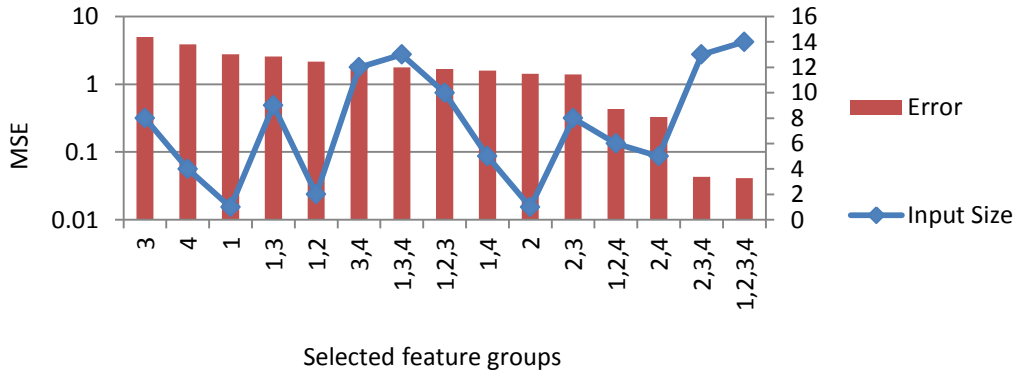


Figure 4.6 Prediction error for neural networks based on different feature groups

Figure 4.6 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 4 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 [20] and [63], 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. New predictions will be made whenever a phase change is detected. Techniques for program phase change detection can be found in [17].

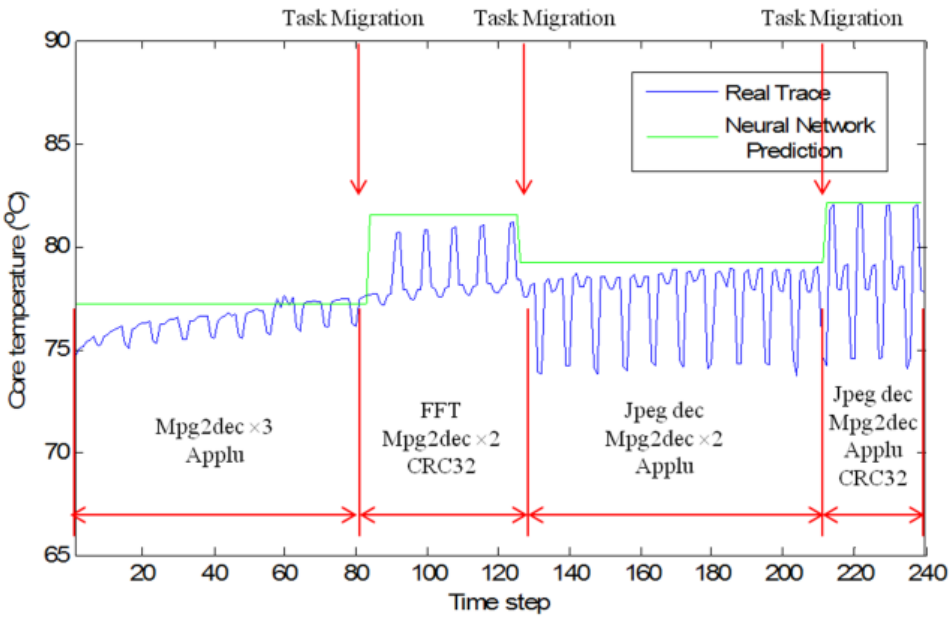


Figure 4.7 The temperature prediction of neural network model

As an example, Figure 4.7 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 seconds 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.

4.4.2 Generalized ARMA Prediction Model

In [20], 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 equation (4.4), 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.

$$y_t + \sum_{i=1}^{p} (a_i y_{t-i}) = e_t + \sum_{i=1}^{q} (c_i e_{t-i}) \tag{4.4}$$

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. Firstly, in a multitasking system where threads start, finish and migrate 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 [39] (i.e. auto-regressive moving average with exogenous inputs.) It is described by equation (4.5), 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.

$$y_t + \sum_{i=1}^{p} (a_i y_{t-i}) = e_t + \sum_{i=1}^{q} (c_i e_{t-i}) + \sum_{i=1}^{r} (b_i u_{t-i}) \quad (4.5)$$

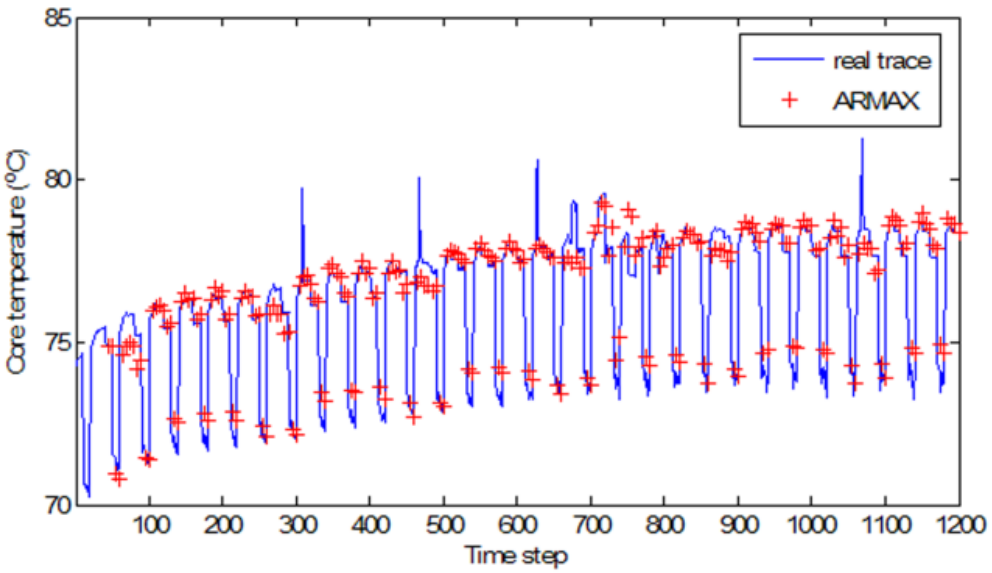


Figure 4.8 The temperature prediction of ARMAX model

Figure 4.8 shows the temperature trace of a PE in a 6x6 many-core system obtained from Hotspot simulation and the temperature prediction made by the ARMAX model. The PE is time multiplexed by 4 tasks. Their execution order is fixed; therefore the temperature trace shows a rough periodic pattern. Similar to reference [20] 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*) [20] is 0.0265.

4.4.3 Comparing the Neural Network Predictor with ARMA Predictor

We compare our neural network based temperature predictor with ARMAX based predictor. Figure 4.9 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 seconds the PE exchanges its high power task with a 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 seconds 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.

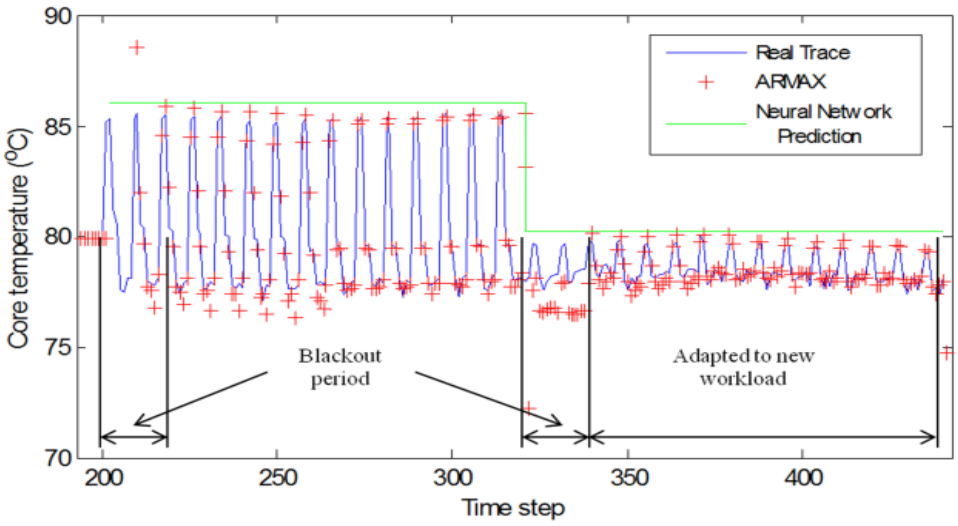


Figure 4.9 The adaption ability of ARMAX model and neural network model

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 4.3. Approximately 200 migrations are randomly generated. Figure 4.10 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.

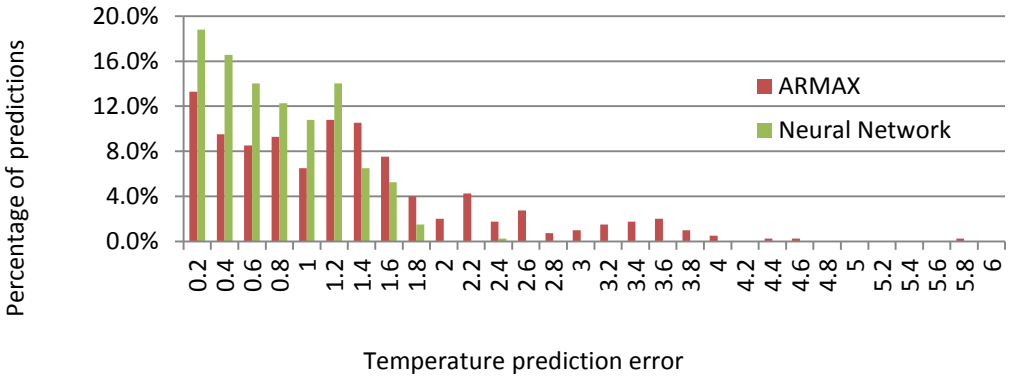


Figure 4.10 Comparison of peak temperature prediction error

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.

4.5 Distributed Task Migration Policy

In this section, we present our distributed task migration policy. Section 4.5.1 discusses the steady state temperature based migration (SSTM) policy. Section 4.5.2 discusses the temperature prediction based migration (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 0 shows how to combine the two migration algorithms together. Finally, section 4.5.4 presents the workload balancing algorithm.

4.5.1 Steady State Temperature Based Task Migration (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 $n = 2\times N+14$ [50]. 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, …, P_n. The relation can be represented by the following equation

$$\boldsymbol{TSS} = \boldsymbol{G}^{-1}\boldsymbol{P} \tag{4.6}$$

where $\boldsymbol{G}^{-1} = [g_{ij}]$ is the inverse of thermal conductance matrix $\boldsymbol{G}$. We simplify equation (4.6) by keeping only the thermal nodes related to the PEs:

$$\begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} \tag{4.7}$$

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 (4.7) 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 equation (4.7), the total steady state temperature change of all processors after task migration can be calculated as:

$$\sum_{k=1}^{N} \Delta T_k = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j \tag{4.8}$$

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 2 multiplications and 1 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, equation (4.8) 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** ($\Delta T_{min} < 0$) swap($\tau_i,\ \tau_j$)
6. **} while** ($\Delta T_{min} < 0$)

4.5.2 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 $\tau_i \in LT_i$
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 (τ_i, τ_j) to the master and exit;
9. Return NULL to the master and exit;

To escape from the above mentioned situation, we further present the Temperature Prediction Based Migration (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 4.4.

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: $S = \{(\tau_i, \tau_j) \mid (\tau_i, \tau_j)$ 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_j} is the minimum

4.5.3 The 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 presented 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.

Similar as many other research works in thermal management of multi-core systems [24], we assume that the peak temperature of each core can be captured by one sensor located in the hottest module, e.g. the register file. However, even if multiple hotspots exist in a core, the presented DTB-M algorithm can still be applied as long as we are able to predict the peak temperature of all the hotspots, and then the same decision process will be carried on based on

the highest temperature. In order to predict multiple temperature readings, we can add more neurons in the output layer of the neural network predictor so that it generates multiple outputs. The feature set should also be modified to include the information such as the switching activities of those hot modules. As the model becomes more complicated, it also needs larger training set to be properly trained.

Finally, the proposed DTB-M is a runtime thermal management technique. It can be used together with any design time thermal optimization techniques such as hot-core and cold-cache interleaving [43].

4.5.4 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 $\left||LT_i| - |LT_j|\right| > nt_{diff}, j \ni N_i$. 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, equation (4.8) 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 $|LT_i| > \left|LT_j\right|$, 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 $|LT_i| < \left|LT_j\right|$, the migration from PE_j to PE_i will start from the task with the lowest power.

4.6 Experimental Results

We implemented a power trace driven behavioral simulator of a many-core system using C++. Hotspot [54] 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 6x6 grids is chosen for our experiments due to the limitation of simulation time. Each core has a size of 4mm x 4mm with silicon layer of 24mm x 24mm. We set the thermal sampling interval of Hotspot to $30\mu s$, in order to speed up the simulation without significantly reducing the accuracy.

We evaluated the presented 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^{o}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_{diff} = 2, t_{slice} = 100ms, $_{T_{diff}}$ = 10^{o}C, δ=0.5^{o}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^{o}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.
- *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 [14]. The average power consumptions and steady state temperatures of each task are summarized in TABLE 4.3. 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.

TABLE 4.3 Average Power and Steady State Temperature of CPU Benchmarks

No.	1	2	3	4	5	6	7	8	9
Benchmarks	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. (^{o}C)	99.42	84.17	82.95	81.42	78.07	77.76	65.56	57.63	56.72

4.6.1 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 4 tasks. The number of each benchmark in the task set follows a specific discrete probability

distribution of its average power consumption. Figure 4.11 shows the 5 different distributions tested in the experiment.

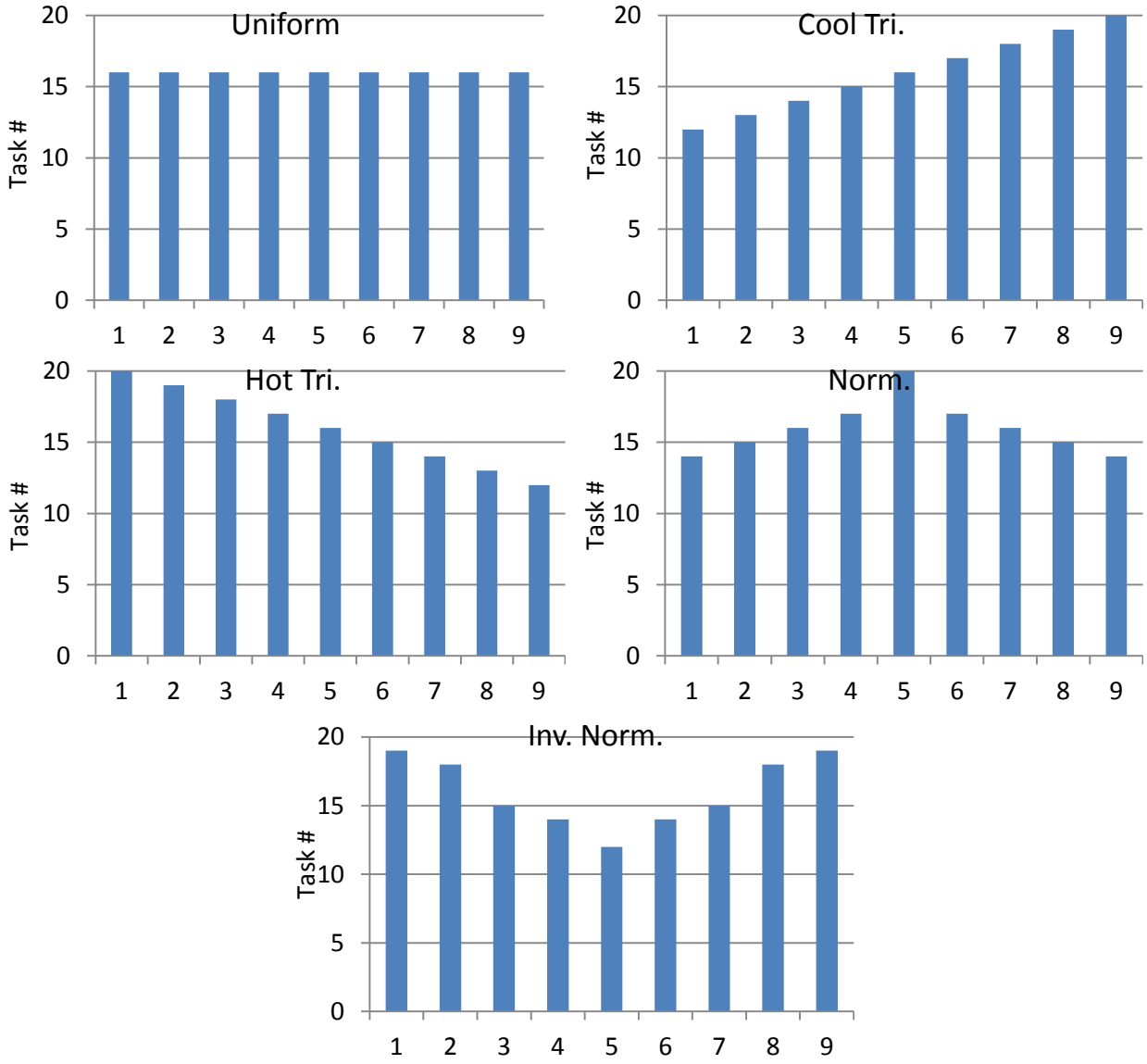


Figure 4.11 Different task set generation probability distribution

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

4.6.2 The 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 4.4 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.

TABLE 4.4 Comparison between ARMAX and Neural Network Model

Workload Distribution	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

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.

4.6.3 The 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 seconds. We simulate both systems for equal length of period and compare their performance.

TABLE 4.5 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

As shown in TABLE 4.5, 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.

4.6.4 The Comparison between SSTM and TPM Policies

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

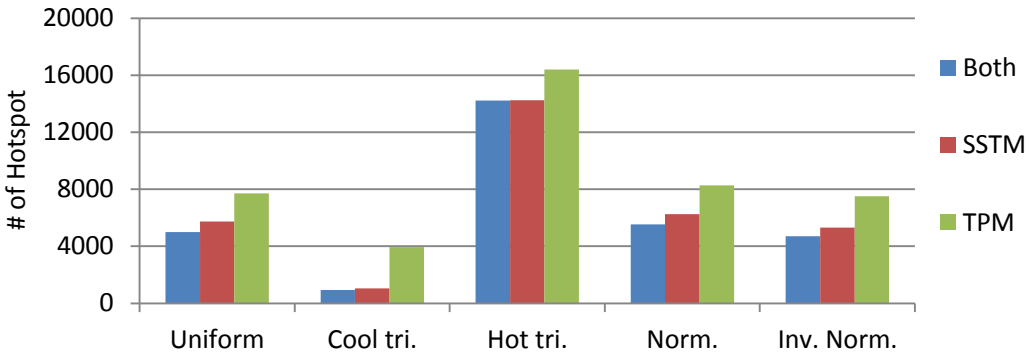


Figure 4.12 Comparison of hotspots

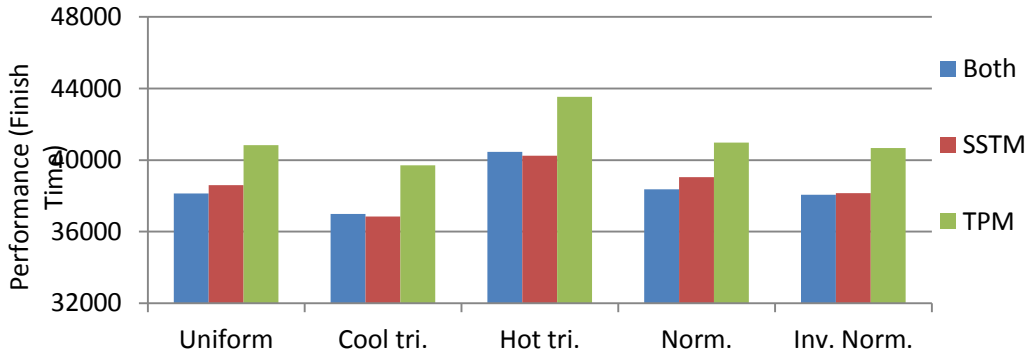


Figure 4.13 Comparison of performance

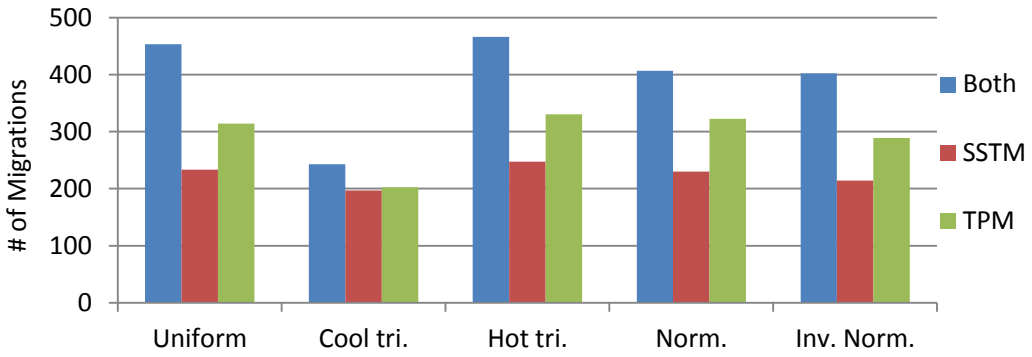


Figure 4.14 Comparison of number of migrations

Figure 4.12 through Figure 4.14 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.

Figure 4.12 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 hotspot reduction than TPM, in DTB-M, we give SSTM higher priority and perform it before the TPM.

4.6.5 The Comparison between Distributed Policy and Global Policy

As we mentioned at the beginning of the chapter, 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.

TABLE 4.6 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%

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 4.6 Comparison between Global and Distributed Policy 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.

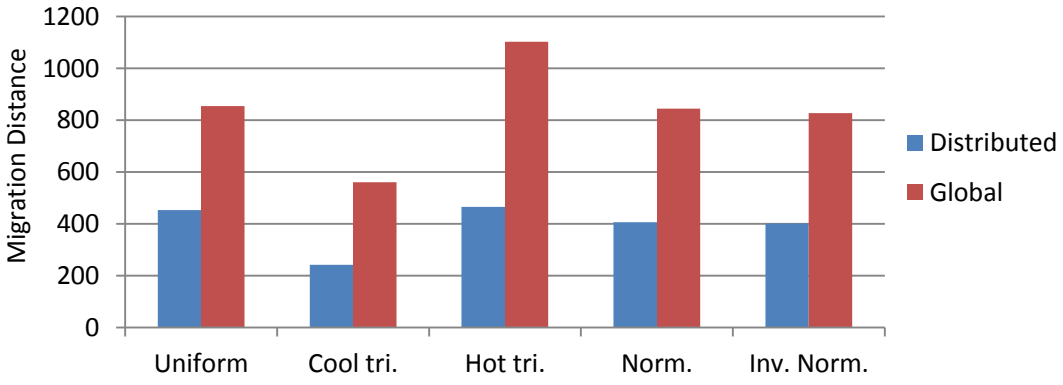


Figure 4.15 Comparison of migration distance

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. Figure 4.15 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.

TABLE 4.7 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

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 4.7 presents the performance of the global policy and the distributed policy for the two extreme cases. Comparing the results in TABLE 4.6 and TABLE 4.7, 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.

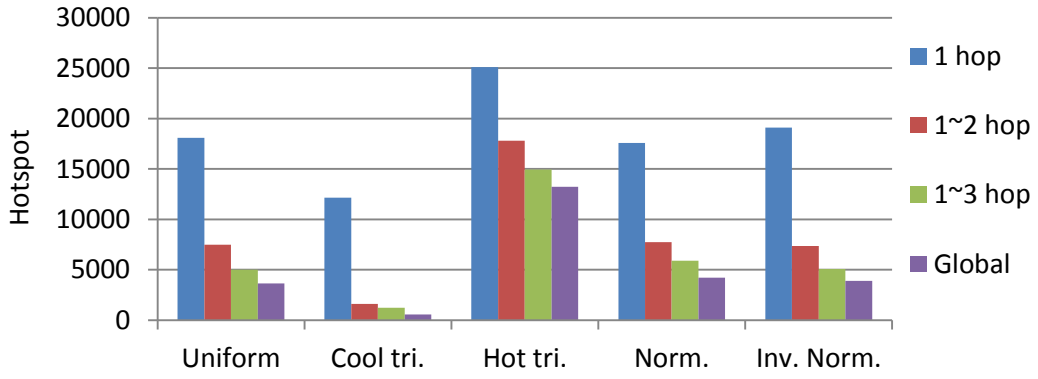


Figure 4.16 Comparison of hotspots between multi-hops distributed policy and global policy

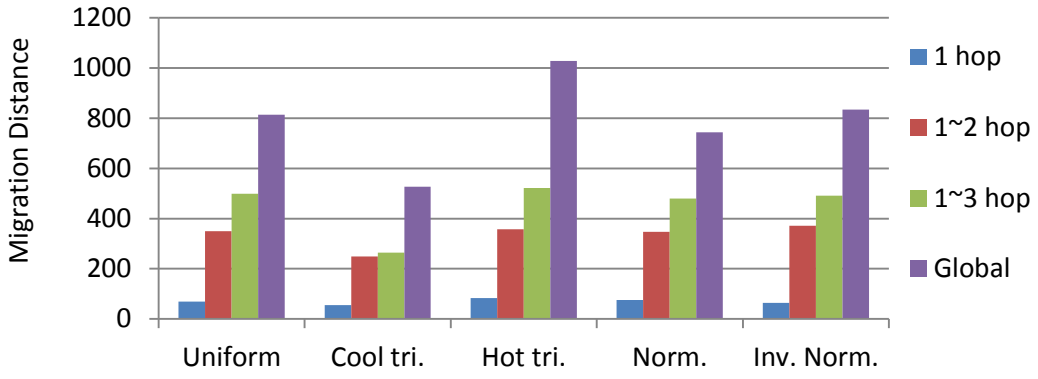


Figure 4.17 Comparison of migration distance between multi-hops distributed policy and global policy

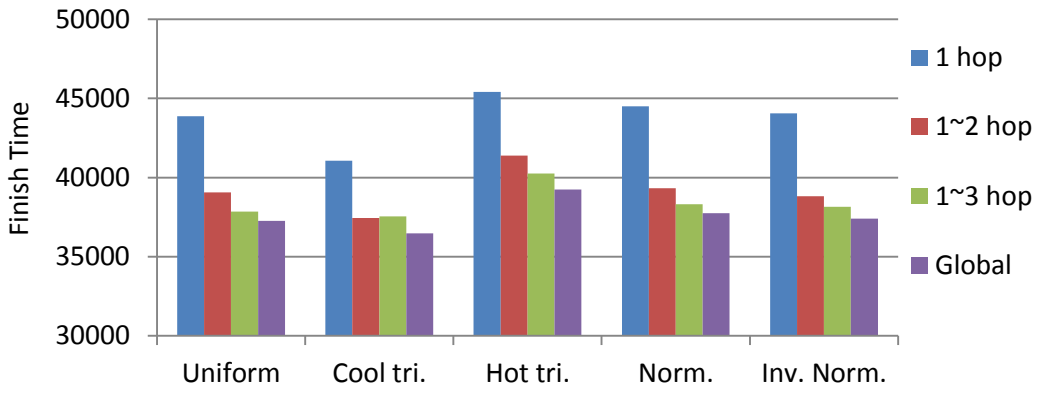


Figure 4.18 Comparison of finish time between multi-hops distributed policy and global policy

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. Figure 4.16 through Figure 4.18 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 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 Figure 4.17, 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.

4.6.6 The Comparison between PTB and DTB-M

In this set of experiments, we compare the DTB-M policy with the *Predictive Thermal Balancing* (*PTB*) policy proposed in [20]. 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 [20] 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 4.8 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%

TABLE 4.8 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, but since they trigger more thermal throttling time, the performance is only slightly better than that of the DTB-M policy for some test cases.

4.6.7 The Impacts of Migration Overhead

In the last experiment, we evaluate the impact of migration overhead to the performance of the overall system. We compare the system using DTB-M to a system without any task migration. When two tasks are exchanged between nearest neighbors, both of them would experience a delay t_{delay}. In this experiment, we vary the value of t_{delay} from 1% of the execution

interval t_{slice} to 100% of t_{slice} and record the finish time for different static workload distributions. The results are shown in Figure 4.19. As we can see, when the t_{delay} is 1% of the t_{slice}, the system using DTB-M is 23.08% faster than the simple system without task migration. This number reduces to 10.34% when t_{delay} is 100% of t_{slice}, which is very unlikely to be true in a real system. The results show that a system that migrates tasks extremely slow still runs faster than a system that does not migrate tasks at all. This is because the simple system that does not perform task migration will be slowed down by lots of thermal throttling events.

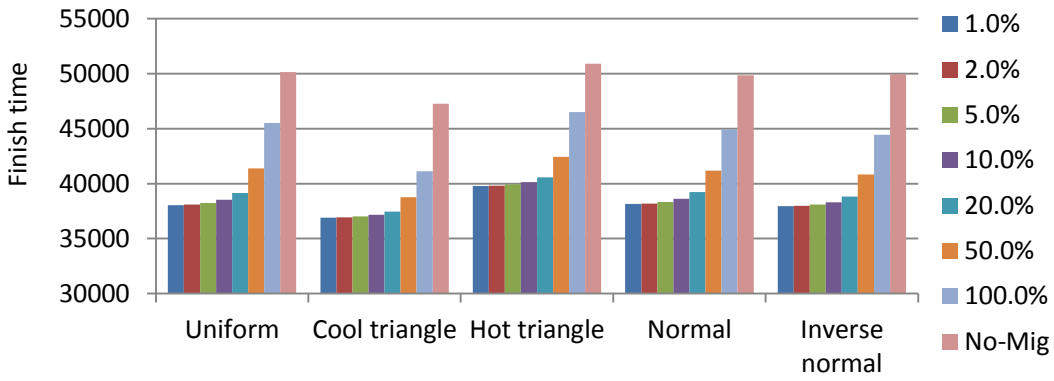


Figure 4.19 Comparison of finish time for different migration overhead

4.7 Chapter Summary

In this chapter, we presented 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 presented 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 hotspot by 29.8% and migration overhead by 80.68% with only 0.98% performance overhead compare to the PTB thermal management.

Chapter 5 Task Allocation for Cooling and Leakage Power Optimization

As shown in the last chapter, even in a homogenous many/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 sufficiently high 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 higher speed to maintain high air flow and strong heat dissipation ability. However, operating at high speed for long time consume more energy and reduce fan life time [11].

In this chapter, we address the impact of task mapping on the overall power consumption of a homogenous multi-core system. The system power consists of three components, dynamic power, static power, and fan power. Due to the homogeneity, different task mappings have little impact on the dynamic power. However, they change 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. Hence they require different optimization techniques. However, as we will show in Section 5.2 that the impact on leakage power from task mapping is negligible if the fan speed is given. 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. Our analysis shows that the overall power can be minimized if a task allocation with minimum peak temperature is adopted together with an intelligent fan speed adjustment technique that finds the optimal tradeoff between fan power and leakage power. Furthermore, 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 further investigate the techniques to search for the task allocation for minimum peak temperature. We formulate the task allocation problem as a zero-one linear programming. Because solving the binary linear programming problem does not scale well for large system, we presented an agent based distributed task migration approach for peak temperature reduction. Our agent based algorithm has good scalability as the number of processors increases. It achieves up to 18% power savings compare to a random mapping policy.

The rest of the chapter is organized as follows: Section 5.1 introduces the many core system model, the system power and thermal model and cooling system model. We formulate the task allocation problem in section 5.2. In Section 5.3 we present the temperature aware distributed task migration framework. Experimental results are reported in Section 5.4. Finally, we conclude the chapter in Section 5.5.

5.1 System Model

5.1.1 Processor Model

In this chapter, we consider a tile-based network-on-chip many-core architecture [26]. Each tile is a processor with dedicated memory and an embedded router. It will also be referred

to as core or processing element (PE) in this chapter. 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*. Note that a pair of cores that are nearest neighbors sometimes does not have to be close to each other geometrically.

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 [24]. 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.

5.1.2 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 $n = 4\times N+12$ [50]. Let TSS_i and P_i 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 because they does not consume any power. Let TSS and P denote vectors of TSS_i and P_i, $1\leq i\leq n$. When the system reaches the steady state, for each thermal node, its temperature is a linear function of power consumptions P_1, P_2, …, P_n. The relation can be represented by the following equation

$$TSS = \boldsymbol{G}^{-1}P \tag{5.1}$$

where $\boldsymbol{G}^{-1} = [g_{ij}]$ is the inverse matrix of thermal conductance matrix $\boldsymbol{G}$. We simplify equation (5.1) by keeping only the thermal nodes related to the PEs:

$$\begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} \tag{5.2}$$

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 consumption P_j of those thermal nodes which are not related with processors (e.g. the thermal nodes on heat sink and heat spreader) does not change. The coefficients g_{ij} and D_i $1 \leq i, j \leq N$ can be obtained by offline analysis. Equation (5.2) shows that the steady state temperature of each PE is a linear function of average power consumptions on other PEs and increasing/decreasing the power consumption of one PE will have an impact on the steady state temperature of all other PEs.

5.1.3 Processor Cooling Model

In this chapter, we assume one standard heat sink and fan cooling system for the entire many-core chip as the configuration of the TILERA TILE64 processor [8]. Our cooling system modeling follows the similar techniques described in the previous works [11][53].

$$P_{fan} \propto {R_{conv}}^{-\frac{3}{\alpha}} \tag{5.3}$$

We next model the relation between the convective resistance and the die temperature. Although the Hotspot [54] 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 as shown in Figure 5.1 [11]. In this model, P_i, C_i, and R_i are the power consumption, thermal conductance and die to package thermal resistance of processor i respectively. R_{conv} is the convective resistance.

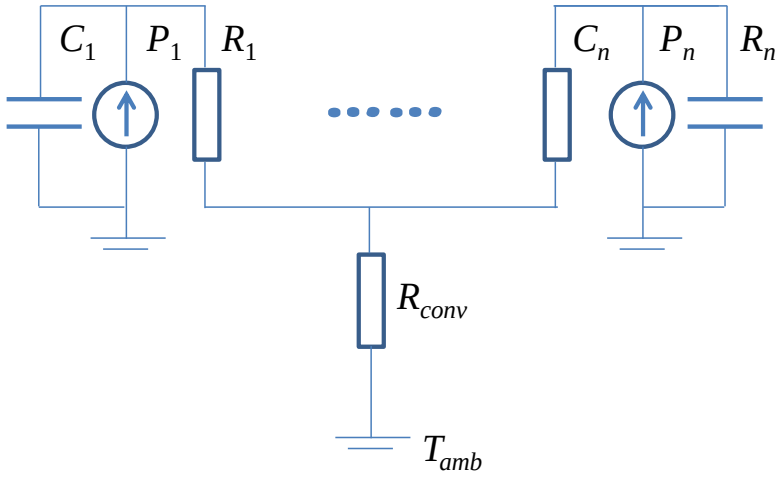


Figure 5.1 Simplified multiprocessor thermal model

Similar to the previous works [53] we are only interested in the temperature at steady state when the system reaches the equilibrium. 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

$$T_i = P_i R_i + R_{conv} \sum_{i=1}^{N} P_i \tag{5.4}$$

where P_i is the power consumption of core i and R_i is the approximation thermal resistance from die to package. 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 5.2 shows that the simulated core temperature and the core temperature predicted by the linear model matches very well.

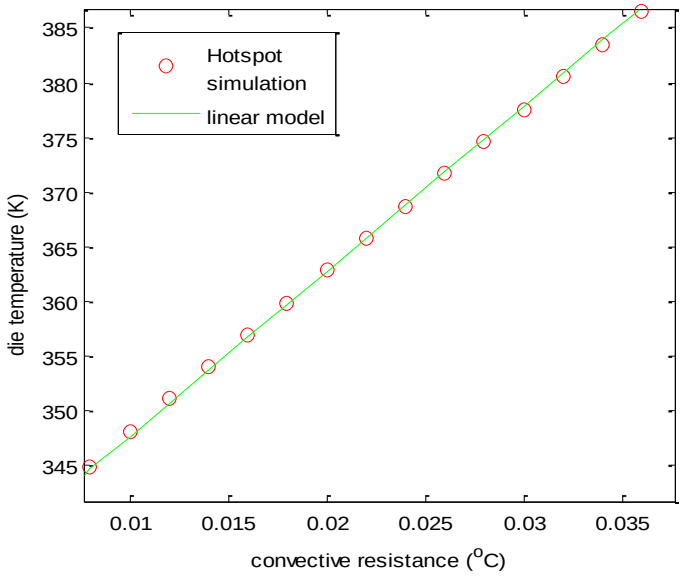


Figure 5.2 Linear approximation of relation between die temperature and convective resistance

5.1.4 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:

$$P_{leak} = A_1 T_d^2 e^{\frac{A_2}{T_d}} + A_3 \tag{5.5}$$

Where A_1, A_2 and A_3 are constants which are dependent on processing technology and supply voltage, T_d is the die temperature. It has been pointed out in [38] that the leakage power can be approximated using a linear model and the resulting error is expected to be less than 5%

for 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 5.3. The green line is the linear approximation while the red line is the original model given by equation (5.5). Figure 5.3 shows that the linear model has very small error compare to the original model in the normal operating range, which is between 60°C and 100°C.

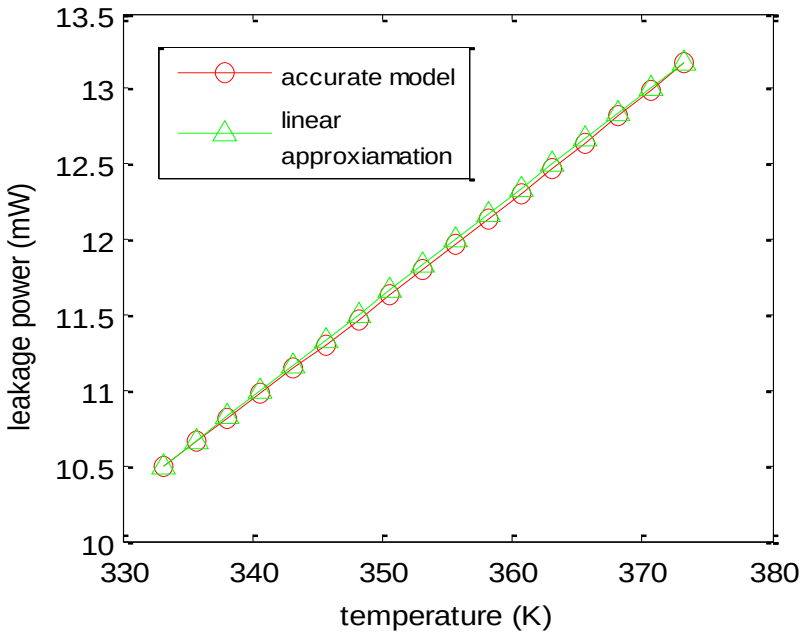


Figure 5.3 Linear approximation of leakage power model

Based on this observation, we approximate the leakage power of the *i*th core using a linear model $P_{leak,i} = aT_{d,i} + b$, 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 $P_{leaktotal} = a\sum T_{d,i} + Nb = NaT_{d-avg} + Nb$, 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 $P_{leaktotal}$ is also a linear function of the convective resistance.

5.2 Problem Formulation and Analysis

In this chapter, 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.

$$P_{total} = P_{dyn} + P_{leak} + P_{fan} \tag{5.6}$$

In a homogenous 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 power consumptions.

To show the relation between task allocation and leakage power consumption, we randomly generated 100 groups of task allocation for a given workload and compare their leakage power consumption on a 36 core chip multiprocessor. The workload consists of 36 tasks whose power consumption varies from 10mW to 20mw (details about workload generation are described in section 5.4.) Figure 5.4 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 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 5.5 (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 scheme 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^{o}C.

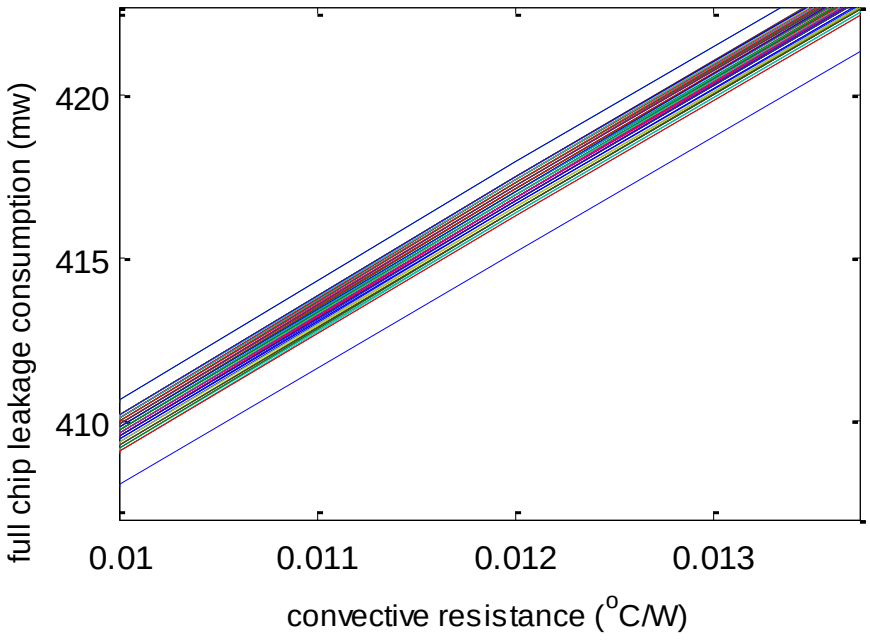
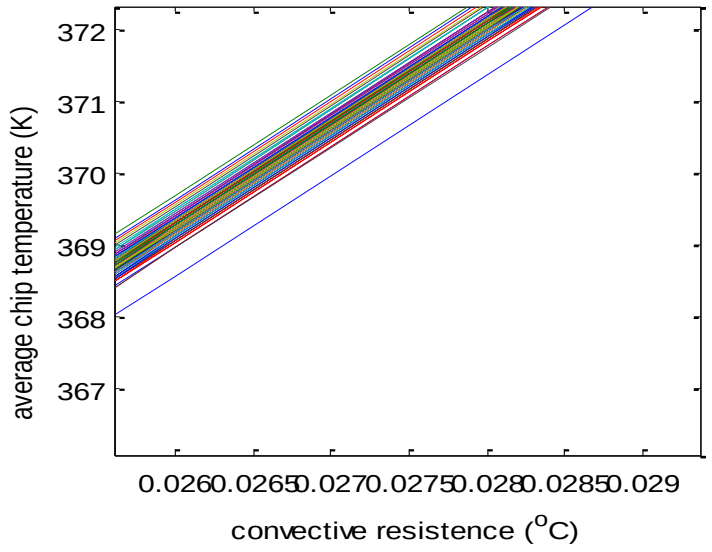


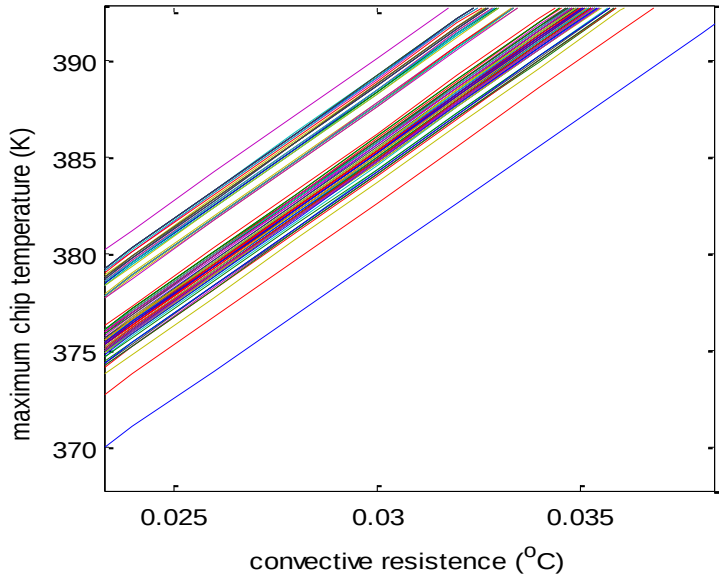
Figure 5.4 The relation between full chip leakage power consumption and different task allocations

Based on the experimental results we have two observations, (1) for a given R_{conv}, the leakage power can be considered to be independent to 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: $P_{leak} = c_1 R_{conv} + c_2$.



a)



b)

Figure 5.5 Comparison of maximum temperature of 3 different task allocations

On the other hand, different task allocation significantly affects 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 5.5 (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^{o}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 that generates the most balanced temperature distribution. A task allocation that generates highly unbalanced temperature distribution will force the cooling fan to work harder (and consumes more power) to keep the peak temperature under the constraint. 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 to 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 on 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, for a given workload, task allocation cannot change the relation between the overall power consumption and the convective resistance.

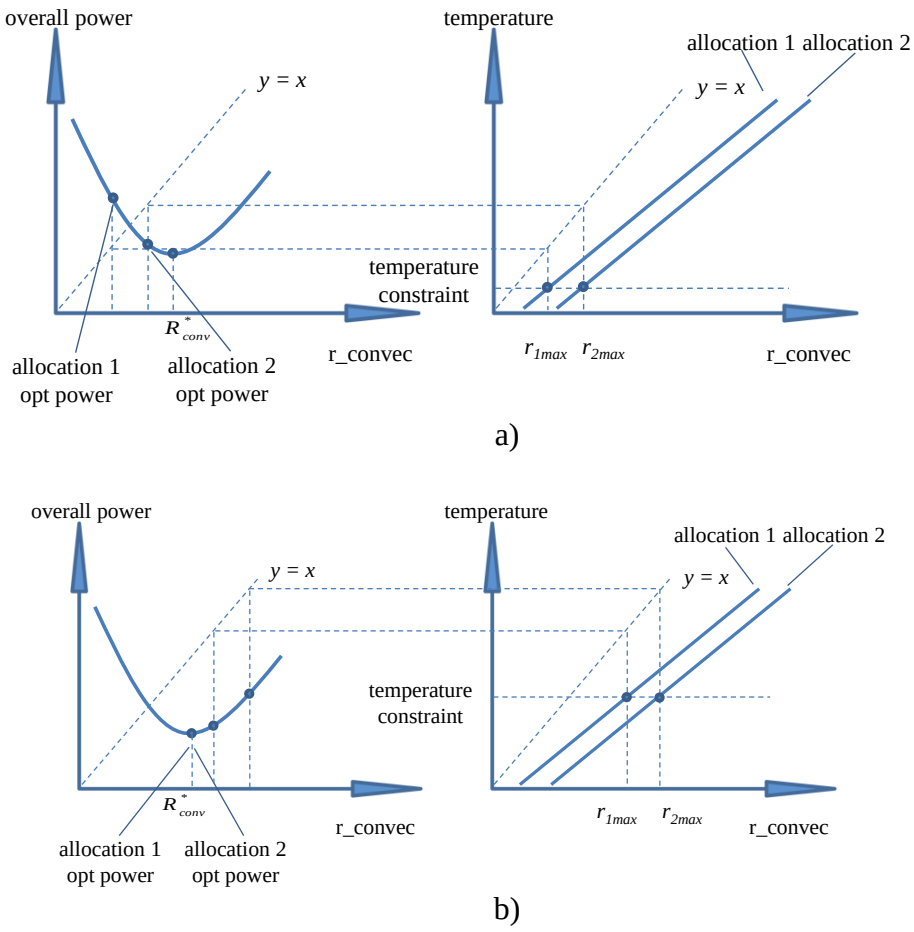


Figure 5.6 The overall power consumption depend on convective resistance

Figure 5.6 shows the overall power consumption and the peak temperature under different task allocations as functions of the convective resistance. Figure 5.6 (a) shows the scenario when the temperature constraint is strict and the convective resistance (i.e. r_1 and r_2) that could 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 minimizes the peak temperature. Figure 5.6 (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}. In this case the leakage power dominates the overall power. If the fan speed is set to exactly satisfy the temperature constraint (i.e. the convective resistance is set to r_1 or r_2) then the best allocation is scheme 1 which has higher peak temperature. We denote the maximum R_{conv} that keeps the peak temperature under constraint as r_{max}. Obviously, any convective resistance that is less than r_{max} can keep the system in safe temperature zone. This includes R^*_{conv}. For both allocation 1 and 2, setting the convective resistance to R^*_{conv} can minimizes the overall power while satisfying the temperature constraint. Because 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 would increase the fan power but could reduce the temperature and leakage power. In this case, power consumption is not sensitive to task allocation. Any allocation scheme whose r_{max} is greater than R^*_{conv} could be used to find the optimal tradeoff point 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.

5.3 Power Optimal Task Allocation

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 point between fan power and leakage power such that the overall power consumption is minimized and the temperature constraint is satisfied. The latter step could be achieved by using feedback control while former step will be discussed in detail in the following sections.

5.3.1 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 $\{\tau_1, \tau_2, \dots, \tau_n\}$ whose power consumption $\{P_1, P_2, \dots, P_n\}$ can be obtained through offline training or online estimation by observing the event counter. We assume the power consumption is a constant for each task, because we are only concerned about the steady state temperature. Here we assume that the core does not support multitasking and the number of tasks is equal to the number of cores. If the number of tasks is less than the number of cores, we can simply add some dummy tasks with 0 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:

$$\min \max_i \left(\sum_{j=1}^{n}\sum_{k=1}^{n} g_{ij} x_{jk} P_k\right) + D_i \tag{5.7}$$

Subject to

$$\sum_{j=1}^{n} x_{jk} = 1, \forall k = 1, \dots, n \tag{5.8}$$

$$\sum_{k=1}^{n} x_{jk} = 1, \forall j = 1, \dots, n \tag{5.9}$$

$$x_{jk} \in \{0, 1\} \tag{5.10}$$

Constraint (5.8) guarantees that a processor is only occupied by one task and constraint (5.9) 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 *i*th core. To see this, we rewrite the equation (5.2) as follows:

$$\begin{pmatrix} T_1 \\ \vdots \\ T_N \end{pmatrix} = \begin{pmatrix} g_{11} & \cdots & g_{1N} \\ \vdots & \ddots & \vdots \\ g_{N1} & \cdots & g_{NN} \end{pmatrix} \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ \vdots & \ddots & \vdots \\ x_{N1} & \cdots & x_{NN} \end{pmatrix} \begin{pmatrix} P_1 \\ \vdots \\ P_N \end{pmatrix} + \begin{pmatrix} D_1 \\ \vdots \\ D_N \end{pmatrix} \tag{5.11}$$

where $X = [x_{ij}]$ is a permutation matrix which assigns the *n* tasks to the processors. Expand the right hand side the equation would give the equation $T_i = a_{ij}x_{jk}P_k$. Then the objective function *min-max(T_i)* is to minimize the maximum temperature among all *n* processors.

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

$$\min\ u \tag{5.12}$$

Subject to:

$$(\sum_{j=1}^{n}\sum_{k=1}^{n} g_{ij}x_{jk}P_k) + D_i \le u, \forall i = 1, \dots, n \tag{5.13}$$

$$\sum_{j=1}^{n} x_{jk} = 1, \forall k = 1, \dots, n \tag{5.14}$$

$$\sum_{k=1}^{n} x_{jk} = 1, \forall j = 1, \dots, n \tag{5.15}$$

$$x_{jk} \in \{0, 1\} \tag{5.16}$$

The above zero-one min-max linear programming is an exact formulation of the task allocation problem. However, it is extremely difficult to solve. 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 [5] on a 3.2GHz Quad core

Xeon processor. Therefore, it is infeasible to use the solution online for power and thermal optimization in the future many-core platform as the core counter could go up to hundreds and thousands [13]. Instead of solving this min-max problem directly, we present the following online heuristic.

5.3.2 Distributed Task Migration

In this section, we presented our 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 implemented as part of the 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 reduces the peak temperature.

Communication protocol

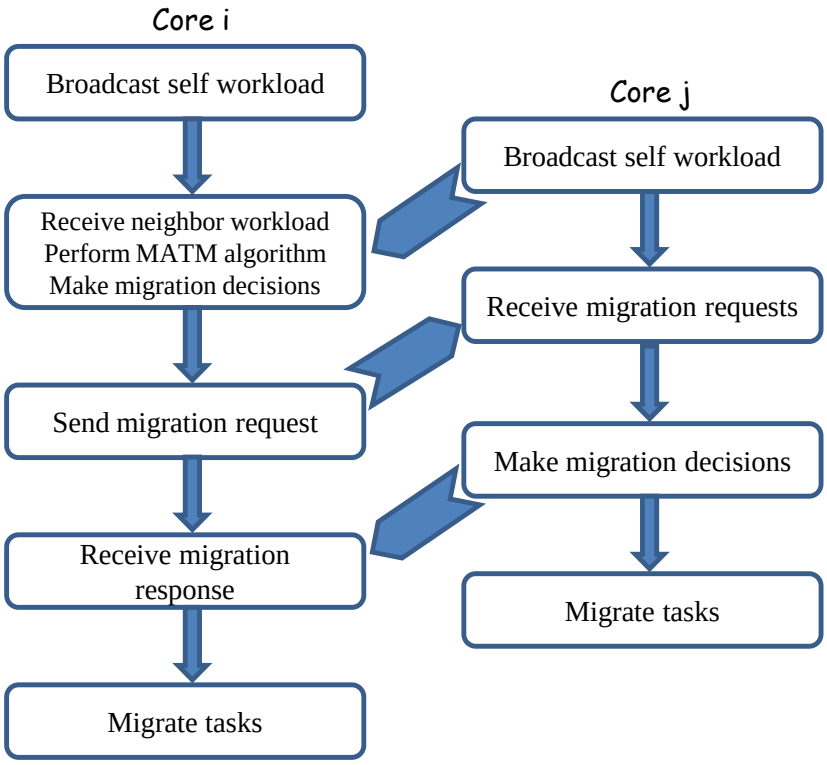


Figure 5.7 Diagram of communication protocol

Each core running a MATM agent can be in two phases: execution phase and communication phase. These two phases are interleaved. During the execution phase the core executes the current computing task while during the communication phase it initiates task migration request to its nearest neighbor or respond to the task migration request from its nearest neighbor. The communication phase can further be divided into four sub-phases: broadcasting self workload to neighboring cores, receiving workload information from neighbors, sending migration requests to neighbors, exchanging tasks with neighbors. Figure 5.7 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 communication phase synchronously in order to communicate to each other. We also refer the communication phase as scheduling interval.

At the beginning of each scheduling interval, an agent on a processor would broadcast its own work load 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 response 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.

MATM distributed migration algorithm

The FDTM algorithm distributes the tasks among processors based on their heat dissipation. It moves high power tasks to processors with strong heat dissipation capability and moves low power tasks to processors with weak heat dissipation capability. 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 equation (5.2) again. Assume that PE_i and PE_j exchange tasks, and their average power consumptions are altered by ΔP_i and ΔP_j respectively. Using equation (5.2), the total die temperature change of all processors in the system after task migration can be calculated as:

$$\sum_{k=1}^{N} \Delta T_k = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j \tag{5.17}$$

where G_i (or G_j) is a parameter that characterizes the heat dissipation ability of processor i (or j),. $G_i = \sum_{m=1}^{N} g_{mi}$, $G_j = \sum_{n=1}^{N} g_{nj}$. The temperature contributed by processor i running task k can be calculated as $G_i P_k$. If $G_i < G_j$, then the temperature contribution made by processor i will be less than the contribution made by processor j when running the same task, which means processor i can dissipate heat better. Thus running high power task on processor i have smaller chance to produce high peak temperature than running high power task on processor j. Therefore, if $G_i < G_j$ and $P_i < P_j$, it is reasonable to switch the tasks on the two processors. This leads to $\sum_{k=1}^{N} \Delta T_k < 0$ in (5.17). In conclusion, 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 lead 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 the following algorithm.

Algorithm 5.1 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

5.4 Experimental Results

The simulation infrastructure and the workload are exactly same as presented in section 4.6.1, so we skip the experimental setup description here.

5.4.1 Fan Power Savings

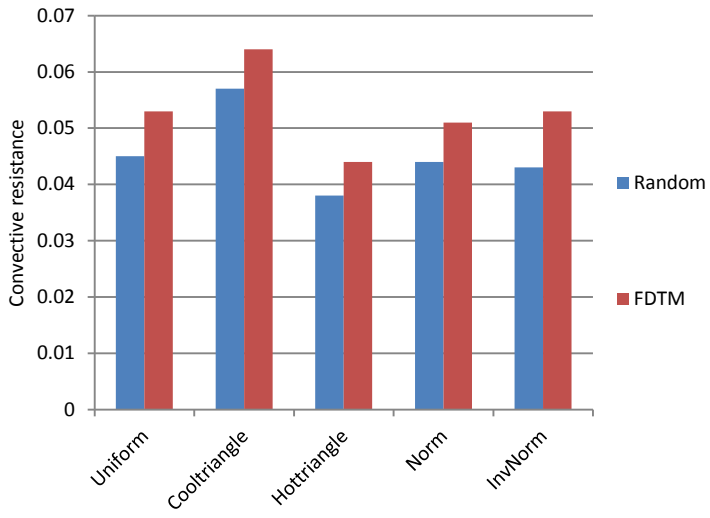


Figure 5.8 Convective resistance comparison between Random allocation and FDTM allocation

Figure 5.8 compares the r_{max} (i.e. the maximum thermal convective resistance that is required to exactly meet the temperature constraint) between the random allocation and MATM based allocation with the temperature constraint setting to 85^{o}C. The results show that, to maintain the whole system under the temperature constraint, the minimum fan speed required by MATM based allocation is 14.5% less than that is required by the random task allocation. The reduced fan speed could bring cubic savings in fan power for the system. And TABLE 5.1 shows the fan power savings of our proposed MATM policy compared to the random allocation

policy. The MATM can achieve an average of 37.2% fan power savings over random allocation while maintains the maximum chip temperature under the thermal constraint.

The reason that the MATM policy can achieve power savings is that through agent negotiation and task migration, tasks can be distributed among processors evenly according to a processor's heat dissipation ability, i.e. high power tasks are moved to cores with stronger heat dissipation ability while lower power tasks are moved to cores with weaker heat dissipation. Therefore the processors' temperatures are distributed more evenly across the chip and the maximum temperature is reduced. The fan can run at a relatively lower speed to guarantee the temperature constraint. Therefore the fan power savings is achieved.

TABLE 5.1 Fan power savings of FDTM compare to the random task allocation

Workload	Uniform	Cool triangle	Hot triangle	Norm	Inv Norm
Fan power savings	38.79%	29.35%	35.58%	35.78%	46.60%

5.4.2 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^{o}C, 85^{o}C and 90^{o}C and compare the power consumption between MATM based task allocation and random allocation. For both systems, optimal tradeoff point between fan power and leakage power will be searched after the system reaches stable state. As shown in TABLE 5.2, MATM based allocation policy could achieve 17.9% overall power savings when the temperature constraint is 80^{o}C. When the

temperature constraint increases to 85°C and 90°C, the power saving reduces to 5.1% and 1.2% respectively.

TABLE 5.2 Overall system power consumption comparison under different temperature constraints

Overall System Power Consumption (mW)			
Temp. Constraint	80°C	85°C	90°C
Random Mapping	1268.9	1110.8	1067.3
FDTM	1076.6	1057	1055.1

The experimental results show a diminishing power savings as the constraint temperature increases from the TABLE 5.2, and task allocation gives large power savings especially when temperature constraint is strict. To understand this, we draw the overall power consumption and fan power consumption against the convective resistance curve in Figure 5.9. When temperature constraint is strict, the convective resistance has to be small to satisfy the constraint. When fan 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 savings. 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 allocation achieves similar overall system power consumptions. 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.

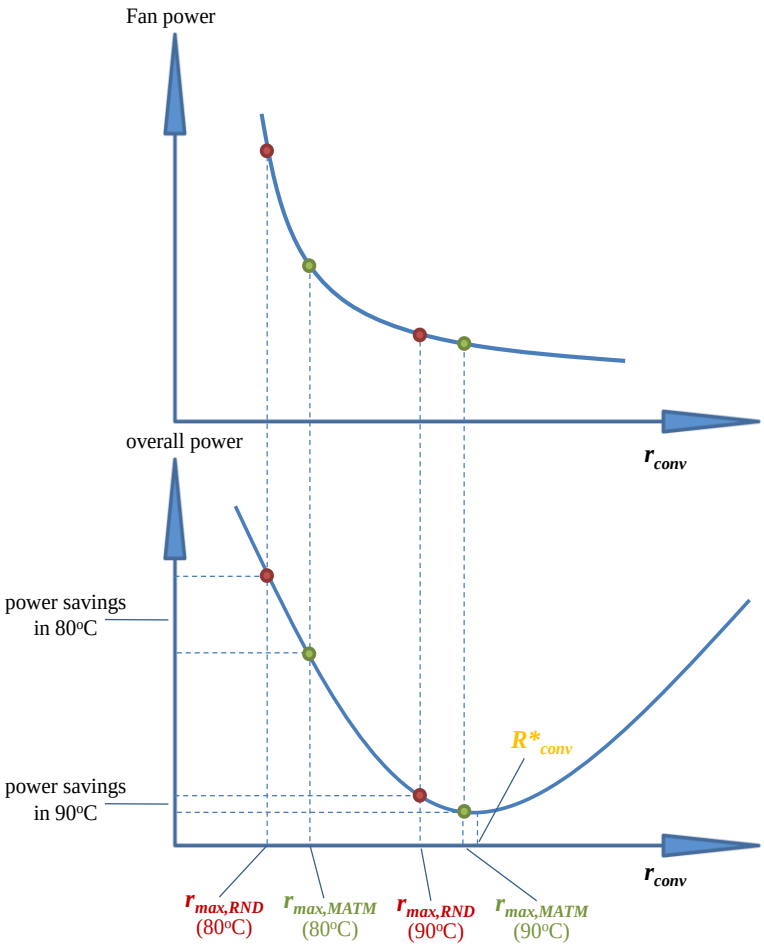


Figure 5.9 Power consumption against convective thermal resistance curve

Figure 5.10 shows each component in the overall system power consumption. The fan power consumption plays an important part in the random allocation when temperature constraint is strict. It accounts for 21.1% of total consumption. When the constraint is relaxed, the share of fan power decreases. The MATM allocation reduces maximum chip temperature and the fan can be maintained in a low speed. Therefore the fan power consumption is small, less than 6% for all temperature constraint. We also notice that the dynamic power stays the same for all constraint 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 the MATM based task allocation has higher leakage power consumption compare to the random allocation. This is because in order to maintain the same maximum chip temperature, the higher fan speed needed for random allocation 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.

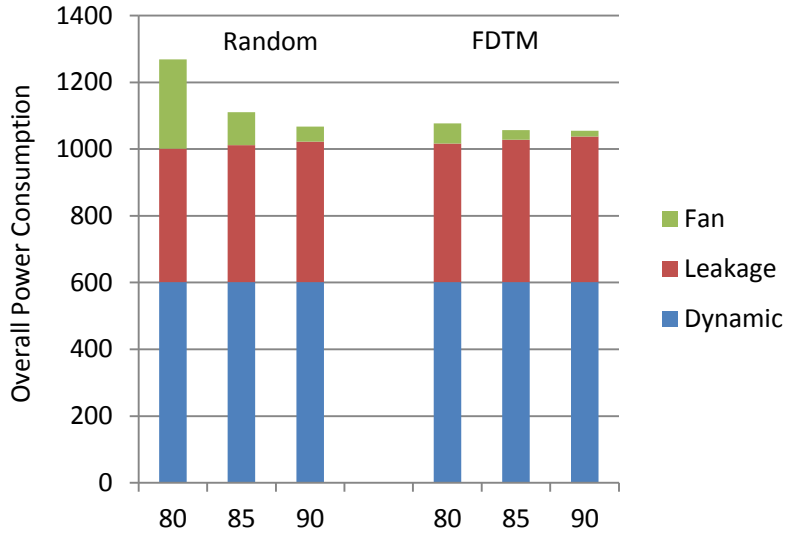


Figure 5.10 The overall power consumption break down

5.5 Chapter Summary

In this chapter, we studied the impact of task mapping on the overall power consumption of a homogenous multi-core system. We formulated the task mapping problem as a zero-one linear programming problem and presented an agent based distributed task migration approach 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.

In this chapter, we did not consider the effects of cooling fan speed on the inlet temperature of servers in a data center environment. Generally, higher cooling fan speed will cause more hot air be circulated back to the inlet. This could reduce the heat removing efficiency of the cooling fan. Reference [46] considers the relation between the inlet temperature and task allocation in a data center. In their work, the inlet temperature of a chassis is modeled as the linear combination of server power consumptions and Air Conditioner temperature. Based on this linear model, they found the optimal task allocation to maximize the Air Conditioner temperature while satisfying the inlet temperature constraint using integer linear programming. However, this work only considers the dynamic power but does not consider fan power and leakage power. How to combine their inlet temperature model with our overall power optimization technique in a data center environment will be our future work.

Chapter 6 Conclusions and Future Directions

Increased power density has set up a “Power Wall” which blocks the micro-processor’s performance improvement, and the clock frequency growth is restricted due to the temperature issues. This is because unmanaged temperature could cause many reliability and performance issues and increase leakage power as well as cooling cost. Dynamic thermal management techniques are designed to tackle the problems and control the chip temperature as well as power consumption. As long as the temperature is regulated, the system reliability can be improved, leakage power can be reduced and cooling system lifetime can be extended significantly.

In this thesis, we have studied several learning based dynamic thermal management techniques where the learning agent monitors the system dynamic and taking appropriate thermal management actions intelligently.

For multimedia applications, the presented learning agent exploits the temporal correlation in this class of application, learns the workload pattern based on the performance counter readings and apply reinforcement learning algorithm to dynamically control the operating frequency of the processor to optimize the performance while ensuring thermal safety. We implemented our learning based DTM policy on a personal computer and tested it using real application. Our experimental results show about 7.53% performance improvement with only about 1.9% thermal violations compare to a thermal management policy that also based on workload phase detection. And compared to another more aggressive policy which has fixed

frequency, our learning policy reduces thermal violation significantly while maintaining the similar run time.

For a many-core system, we presented a distributed thermal management framework where 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 presented 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.

Finally, we studied the impact of task mapping on the leakage power and cooling fan power consumption of a homogenous many-core system. We formulated the task mapping problem as a zero-one linear programming problem and presented an agent based distributed task migration approach to solve this problem. Our agent based algorithm has good scalability as the number of processors increases. The presented algorithm can achieve an average of 37.2% fan power savings over random allocation while maintains the maximum chip temperature under the thermal constraint and it could achieve 17.9% overall power savings when the temperature

constraint is 80^{o}C. When the temperature constraint increases to 85^{o}C and 90^{o}C, the power saving reduces to 5.1% and 1.2% respectively.

6.1 Future Directions

Dynamic thermal management is still an ongoing and active research area. New technologies and computing platforms impose new challenges to existing DTM techniques. One area that has not been well explored is thermal management of 3-D integrated circuits [21]. 3-D IC is designed to tackle the large delay and high power associated with long interconnects. However this new technology escalates the chip power density which has already been a severe issue in 2-D design. Techniques work for 2-D platform might not be effective in 3-D platform, for example, the forced-convection heat sink cooling solution could not work well in 3-D cases. New thermal modeling tools, novel thermal management techniques need to be investigated for this new platform.

In addition, new integration technology brings the opportunities of integrating unconventional cooling solutions on the chip. For example, the authors in [12] investigate the potential cooling power savings in a data center by inserting a Thermal electric (TEC) layer between the silicon die layer and the heat spreader layer for CPUs. The TEC layer works as a high-efficiency heat pump. When configured at appropriate size, TEC is very effective to pump heat from its hot side to its cold side. Stacking the TEC layer on top of the silicon could quickly remove the heat generated from the die and reduce the chip temperature. Therefore, the cooling power consumed by other units like cooling fan and CRAC can be saved. However, the TEC itself needs to consume power. Apparently, there is a trade-off between the power consumption by TEC and other cooling units. The authors provide a detailed hierarchical power/thermal model for different components in a data center including silicon die, TEC layer, heat spreader

layer, heat sink layer, cooling fan and finally CRAC. Their initial results show that, with the TEC layer, the data center CRAC is able to increase its operating temperature from 288K to 294K, which is equal to 24% power savings, without compromising the chip lifetime and reliability. Many other research works also focus on applying liquid cooling techniques onto the chip through the microchannels.

While the concept of dynamic thermal management was initially proposed to reduce the cooling cost, the emerging packaging and cooling techniques will also effective relive the stress of DTM. Many of the modern cooling techniques are now providing control knobs that could tradeoff the cooling efficiency with energy cost, for example, adjustable fan speed, selective turning on/off the TEC units, and adjustable liquid flow rate etc. As we can imagine, the most effective DTM technique should have a holistic view of the entire system, which consists of the software workload activity, the configuration of the integrated circuits, as well as the status of the cooling system; and manage the whole system. This leads to many open questions, including how to model the effects of these new cooling solutions on the heat generated by the silicon, how to efficiently monitor and control the cooling system during runtime, and how to utilize its potential to design management algorithm to achieve power/energy savings, to improve performance and reliability. We believe that research in these areas will extend the potential of thermal management and eventually lead to better, faster, and more reliable integrated system.

Bibliography

[1] “AMDPowerNow”, http://www.amd.com/us/products/technologies/powermanagement/Pages/power-management.aspx

[2] “Enhanced Intel SpeedStep® Technology - How To Document”, http://www.intel.com/cd/channel/reseller/asmo-na/eng/203838.htm

[3] Failure Mechanisms and Models for Semiconductor Devices, JEDEC Publication JEP122C. http://www.jedec.org.

[4] International Technology Road Map for Semiconductor 2011 Edition System Drivers. Available: http://www.itrs.net/Links/2011ITRS/2011Chapters/2011SysDrivers.pdf

[5] http://lpsolve.sourceforge.net/5.5/

[6] MediaBench: http://euler.slu.edu/~fritts/mediabench/

[7] Perfmon2: http://perfmon2.sourceforge.net/pfmon_usersguide.html

[8] Tile Processor Architecture: Technology Brief. Available: http://www.tilera.com/pdf/ProductBrief_TileArchitecture_Web_v4.pdf.

[9] I. Aleksander, H. Morton, *An Introduction to Neural Computing*. International Thomson Computer Press, 1995.

[10] R. Ayoub, and T. Rosing, "Predict and act: dynamic thermal management for multi-core processors," In *Proceedings of International Symposium on Low power Electronics and Design*, pages 99 – 104, 2009.

[11] R. Ayoub, S. Sharifi and T. Rosing, "GentleCool: Cooling Aware Proactive Workload Scheduling in Multi-Machine Systems," In *Proceedings of Design Automation and Test in Europe*, pages 295 – 298, 2010.

[12] S. Biswas, M. Tiwari, T. Sherwood, L. Theogarajan, and F.T. Chong, "Fighting fire with fire: modeling the datacenter-scale effects of targeted superlattice thermal management," In *Proceeding of international symposium on Computer architecture*, pages 331 – 340, 2011.

[13] S. Borkar, "Thousand Core Chips – A Technology Perspective," In *Proceedings of Design Automation Conference*, pages 746 – 749, 2007.

[14] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A Framework for Architectural Level Power Analysis and Optimizations," in Proceedings of International Symposium Computer Architecture, pages 83 – 94, 2000.

[15] X. Chen, C. Xu, R. Dick, Z. Mao, "Performance and Power Modeling in a Multi-Programmed Multi-Core Environment," in *Proceedings of Design Automation Conference*, pages 813 – 818, 2010.

[16] J. Choi, C. Cher, H. Franke, H. Hamann, A. Weger and P. Bose, "Thermal Aware Task Scheduling at the System Software Level," in *Proceedings of International Symposium on Low Power Electronics and Design*, pages 213 – 218, 2007.

[17] R. Cochran, and S. Reda, "Consistent Runtime and Thermal Prediction and Control Through Workload Phase Detection," In *Proceedings of Design Automation Conference*, pages 62 – 67, 2010.

[18] A. Coskun, T. Rosing, and K. Whisnant, "Temperature aware task scheduling in MPSoCs," in *Proceedings of Design Automation and Test in Europe*, pages 1659 – 1664, 2007.

[19] A. Coskun, T. Rosing, and K. Gross, "Proactive temperature balancing for low cost thermal management in MPSoCs," In *Proceedings of International Conference on Computer-Aided Design*, pages 250-257, 2008.

[20] A. Coskun, T. Rosing, and K. Gross, "Utilizing predictors for efficient thermal management in multiprocessor SoCs," In *IEEE Transaction on Computer Aided Design on Integrated Circuits and System*, vol. 28, no. 10, pp. 1503–1516, 2009.

[21] A. Coskun, J. Ayala, D. Atinza, T. Rosing, and Y. Leblebici, "Dynamic Thermal Management in 3D Multicore Architectures," In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1410-1415, 2009.

[22] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of Design Automation Conference*, pages 684 – 689, 2001.

[23] G. Dhiman, and T. Rosing, "System-Level Power Management Using Online Learning," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 676-689, 2009.

[24] J. Donald and M. Martonosi "Techniques for Multicore thermal Management: classification and new exploration," In *Proceedings of International Symposium on Computer Architecture*, pages 78 – 88, 2006.

[25] T. Ebi, M. Al Faruque, and J. Henkel, "TAPE: Thermal-aware agent based power economy for multi/many-core architectures," In *Proceedings of International Conference on Computer Aided Design*, pages 302–309, 2009.

[26] Y. Ge, P. Malani, and Q. Qiu, "Distributed Task Migration for Thermal Management in Many-core Systems," In *Proceedings of Design Automation Conference (DAC)*, pages 579 – 584, 2010.

[27] H. Shen and Qinru Qiu, "Learning Based DVFS for Simultaneous Temperature, Performance and Energy Management," in *Proceesings of International Symposium on Quality Electronic Design*, 2012.

[28] 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 *Proceedings of International Solid-State Circuits Conference*, pages 108 – 109, 2010.

[29] J. Hu, R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proceedings of Asia and South Pacific Design Automation Conference*, pages 233 – 239, 2003.

[30] W. Huang, M. Stan, K. Sankaranarayanan, R. Ribando and K. Skadron, "Many-Core Design from a Thermal Perspective," in *Proceedings of Design Automation Conference*, pages 746 – 749, 2008.

[31] R. Jayaseelan and T. Mitra, "Dynamic Thermal Management via Architectural Adaption," *In Proceedings of Design Automation Conference*, pages 484-489, 2009.

[32] H. Jung, P. Rong and M. Pedram, "Stochastic Modeling of a Thermally-Managed Multi-Core System," *In Proceedings of Design Automation Conference*, page 728 – 733, June 2008.

[33] P. Kongetira, K. Aingaran, and K. Olukotun, " Niagara: A 32-way multithreaded SPARC processor," in *IEEE Micro*, Vol. 25, Issue 2, pages 21–29, 2005.

[34] E. Kursun, C. Cher, A. Buyuktosunoglu, and P. Bose, "Investigating the Effects of Task Scheduling on Thermal Behavior," In *Third Workshop on Temperature-Aware Computer Systems*, 2006.

[35] C. Lefurgy, K. Rajamani, F. Rawson, W. Felter, M. Kistler and T. Keller, "Energy Management for Commercial Servers," In *IEEE Computer*, Vol. 36, Issue 12, pages 39 – 48, 2003.

[36] W. Liao, L. He, and K. Lepak, "Temperature and supply voltage aware performance and power modeling at microarchitecture level," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 24, Issue 7, pages1042 – 1053, 2005.

[37] S. Liu, J. zhang, Q. Wu, and Q. Qiu, "Thermal-Aware Job Allocation and Scheduling for Three Dimensional Chip Multiprocessor," *In Proceedings International Symposium on Quality Electronics Design (ISQED)*, pages 390 – 398, 2010.

[38] Y. Liu, R. P. Dick, L. Shang and H. Yang, "Accurate temperature-dependent integrated circuit leakage power estimation is easy," In *Proceedings of the conference on Design, automation and test in Europe*, pages 1526 – 1531, 2007.

[39] L. Ljung, "System Identification: Theory for the User (2nd Edition)", Upper Saddle River, NJ, Prentice-Hal PTR, 1999.

[40] R. Love, "Linux Kernel Development," *Addison-Wesley Professional*, 2010.

[41] R. Marculescu, U. Ogras, L. Peh, N. Jerger, and Y. Hoskote, "Outstanding Research Problems in NoC Design: System, Microarchitecture, and Circuit Perspectives," In *IEEE Transaction on Computer -Aided Design Integrated Circuits and System*, vol.28, no. 1, pages 3 – 21, 2009.

[42] P. Michaud, A. Seznec, D. Fetis, Y. Sazeides and T. Constantinou, "A study of thread migration in temperature-constrained multicores," In *ACM Transaction of Architecture Code Optimization*, vol.4, no. 2, pp. 9-1 - 9-28, 2007.

[43] K. Modzelewski, J. Miller and A. Belay, "A unified operating system for Clouds and Manycore: fos", In *MIT-CSAIL-TR-2009-059*, 2009.

[44] J. Moorey, J. Chasey, P. Ranganathanz and R. Sharma, "Making Scheduling Cool: Temperature-AwareWorkload Placement in Data Centers," in *Proceedings of USENIX Annual Technical Conference*, pages 5-18, Apr. 2005.

[45] 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 *Proceedings of Design Automation and Test in Europe (DATE)*, pages 734 – 739, 2008.

[46] 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 *Proceedings of international conference on Hardware/software codesign and system synthesis (CODES+ISSS)*, pages 111 – 116, 2007.

[47] V. Nollet, T. Marescaux and D. Verkest, "Operating System Controlled Network on Chip," In *Proceedings of Design Automation Conference* (*DAC*), pages 256 – 259, 2004.

[48] E. Pakbaznia, M. Ghasemazar, and M. Pedram,"Temperature Aware Dynamic Resource Provisioning in a Power Optimized Datacenter," In *Proceedings of Design Automation and Test in Europe*, pages 124 – 129, 2010.

[49] V. Pamula and K. Chakrabarty "Cooling of Integrated Circuits Using Droplet-Based Microfluidics," In *Proceedings of ACM Great Lakes symposium on VLSI*, pages 84 – 87, 2003.

[50] R. Rao and S. Vrudhula, "Fast and accurate prediction of the steady state throughput of multi-core processors under thermal constraints," in *IEEE Transaction Computer-Aided Design Integrated Circuits System*, vol. 28, no.10, pages 1559-1572, Oct. 2009.

[51] L. Shang, L. Peh, A. Kumar, and N. Jha, "Thermal Modeling, Characterization and Management of On-chip Networks," In *International Symposium on Microarchitecture*, pages 67 – 78, 2004.

[52] S. Sharifi, A. Coskun, and T. Rosing "Hybrid Dynamic Energy and Thermal Management in Heterogeneous Embedded Multiprocessor", In *Proceedings of Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 873 – 878, 2010.

[53] D. Shin, N. Chang, J. Choi, S. Chung and E. Chung, "Energy-Optimal Dynamic Thermal Management for Green Computing," In *Proceedings of International Conference on Computer-Aided Design*, pages 652-657, 2009.

[54] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy and D. Tarjan, "Temperature-Aware Microarchitecture: Modeling and Implementation," In *ACM Transaction on Architecture and Code Optimization*, Vol. 1, Issue 1, pages 94 – 125, 2004.

[55] J. Srinivasan and S. Adve, "Predictive dynamic thermal management for multimedia applications," In *Proceedings of International Conference on Supercomputing*, pages 109 – 120, 2003.

[56] R. Sutton and A. Barto, "Reinforcement Learning: An Introduction", *MIT Press*, 1998.

[57] P. Tan, M. Steinbach, and V. Kumar, "Introduction to Data Mining", *Addison-Wesley*, 2005.

[58] Y. Tan, W. Liu and Q. Qiu, "Adaptive Power Management Using Reinforcement Learning," In *Proceedings of International Conference on Computer-Aided Design*, pages 461 – 467, 2009.

[59] Q. Tang, S. Gupta and G. Varsamopoulos, "Energy-Efficient, Thermal-Aware Task Scheduling for Homogeneous, High Performance Computing Data Centers: A Cyber-

Physical Approach," In *IEEE Transaction on Parallel and Distributed System*, vol. 19, issue 11, pages 1458-1472, Nov. 2008.

[60] 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 65nm CMOS," In *Proceedings of International Solid-State Circuits Conference*, pages 98 – 589, 2007.

[61] Y. Wang, K. Ma, and X. Wang. "Temperature-constrained power control for chip multiprocessors with online model estimation," In *Proceedings of International Symposium on Computer Architecture*, pages 314 – 324, 2009.

[62] J. Wawrzynek, D. Patterson, M. Oskin, S. Lu, C. Kozyrakis, J. Hoe, D. Chiou, and K. Asanovic, "RAMP: Research Accelerator for Multiple Processors," In *IEEE Micro*, pages 46 – 57, 2007.

[63] I. Yeo, C. Liu, amd E. Kim, "Predictive Dynamic Thermal Management for Multicore Systems," In *Proceedings of Design Automation Conference*, pages 734 – 739, 2008.

[64] I. Yeo, E. Kim, "Hybrid Dynamic Thermal Management Based on Statistical Characteristics of Multimedia Applications," In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 321 – 326, 2008.

[65] F. Zanini, D. Atienza, and G. De Micheli, "A Control Theory Approach for Thermal Balancing of MPSoC," in *Proceedings of Asia and South Pacific Design Automation Conference* (*ASPDAC*), pages 37 – 42, 2009.

[66] Y. Zhang, D. Parikh, K. Sankaranarayanan, K. Skadron, and M. Stan. "Hotleakage: a temperature-aware model of subthreshold and gate leakage for architects," *University of Virginia Dept. of Computer Science Technical Report*, 2003.

[67] S. Zhang and K. Chatha, "Thermal aware task sequencing on embedded processors," In *Proceedings of Design Automation Conference*, pages 585 – 590, 2010.

Vita

Yang Ge was born in Hangzhou, Zhejiang, China on July 29th 1985, the son of Haitao Ge and Mingyuan Chen. After completing his work at Hangzhou No.2 Middle School, he went to Zhejiang University, China, where he studied telecommunication engineering and received his Bachelor of Science in May 2007. He received his Master of Science degree in Electrical and Computer Engineering from Binghamton University in 2009. He entered the Graduate School at Syracuse University in 2011.