Distributed Task Migration for Thermal Management in Many-core Systems

Yang Ge, Parth Malani, Qinru Qiu
Department of Electrical and Computer Engineering
Binghamton University, State University of New York
{yge2, pmalani1, qqiu}@binghamton.edu

ABSTRACT

In the deep submicron era, thermal hot spots and large temperature gradients significantly impact system reliability, performance, cost and leakage power. As the system complexity increases, it is more and more difficult to perform thermal management in a centralized manner because of state explosion and the overhead of monitoring the entire chip. In this paper, we propose a framework for distributed thermal management for 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/exchange. By choosing only those migration requests that will result balanced workload without generating thermal emergency, the proposed framework maintains workload balance across the system and avoids unnecessary migration. Experimental results show that, compared with existing proactive task migration technique, our approach generates less hotspots and smoother thermal gradient with less migration overhead and higher processing throughput.

Categories and Subject Descriptors

C.4 [**Perfomance of Systems**]: Reliability, availability, and serviceability

General Terms

Algorithms, Management, Performance

Keywords

Dynamic thermal management, distributed control, prediction

1. INTRODUCTION

The *Multiprocessor System-on-Chip* (MPSoC) is becoming a major system design platform for general purpose and real-time applications, due to its advantages in low design cost and high performance. With the scaling of CMOS devices, this technology is progressing from the *multi-core* era to the *many-core* era [1]. An example of such system is the 80 tile network-on-chip that has been fabricated and tested by Intel [10]. However, the increasing chip complexity and power envelope elevate peak temperatures of chip and imbalance the thermal gradient.

Raised peak temperatures reduce life-time of the core, deteriorate its performance, affect the reliability [9] and increase the cooling cost. Leakage power increases with rising temperature which in turn increases temperature of a transistor resulting in adverse positive feedback effect called thermal runaway [9]. When mapped on many-core system, diverse workload of applications may lead to power and temperature imbalance within different cores. Such temporal and spatial variation in temperature creates local temperature maxima on the chip called hotspot [5][9]. Rising temperatures cause Dynamic Thermal Management (DTM) events such as core throttling or stalling which hit the performance [1]. An excessive spatial temperature variation, which is also referred to as thermal gradient, increases clock skews and decreases performance and reliability.

Considerable work has been done focusing thermal management on multicore systems. Modern day microprocessers handle thermal emergencies through various DTM mechanisms. Techniques at microarchitecture level has been well explored [1][9]. Voltage scaling and scheduling can be combined to leverage the temperature reduction on MPSoCs [5][8].In a many-core system, the heat dissipation capability differs from processor to processor. In [14] an algorithm is proposed to map and schedule tasks based on the thermal conductivity of different processors.

Proactive thermal management based on runtime task migration has been proposed in reference [4] and [11]. Both of them predict the future temperature as a projection of the history temperature trace. Although these predictive models are very accurate in some circumstances, they have some limitations. First of all, both models have to be updated and adjusted at runtime. This could incur adaption overhead. Secondly, both models predict the future temperature solely from the temperature history. For a system with frequent task migrations, history temperature 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. Furthermore, a prediction model works well for an application in one core might not remain to be effective after this application has been migrated to another core because different cores can display different thermal characteristics due to their locations or heat dissipation abilities. Finally, the migration policies proposed in [4] and [11] does not try to maintain a balanced workload among processors. Therefore, they work efficiently only when the number of tasks is less than the number of cores.

Most of these 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 [6].

In this paper we propose a framework of distributed thermal management where balanced thermal profile can be achieved by proactive thermal throttling as well as thermal-aware task migrations among neighboring cores. The framework has a low

DAC'10, June 13–18, 2010, Anaheim, California, USA.

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. We refer to the proposed technique as *distributed thermal balancing migration* (DTB-M) as it aims at balancing the workload and temperature of the processors simultaneously. In this work, we assume that the average power consumption of each task is known. This information can be obtained through offline characterization or online power estimation by observing the event counters.

A neural network based peak temperature predictor is also proposed in this paper. 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 following summarizes the key differences between the proposed thermal management framework and the previous works.

(1) No centralized controller is required in this framework. The distributed thermal management agent communicates and exchanges tasks with its nearest neighbor. Therefore, the communication cost and migration overhead for each core does not increase when the number of cores in the system increases.

(2) The neural network based peak temperature predictor works robustly when the workload changes, which usually happens after task migration/exchange.

(3) Compared to the existing proactive thermal-aware task-migration, the proposed migration policy does not only reduce hot spots and thermal gradient, but also maintains a balanced workload and hence better performance. Experimental results show that the DTB-M has 66.79% less hot spots and 40.21% higher performance than the PDTM proposed in [11]. Furthermore, the DTB-M also has much lower migration overhead. The number of migrations is reduced by 33.84% while the overall migration distance is reduced by 70.7%.

The rest of the paper is organized as follows: Section 2 gives the semantics of the underlying many core system and the application model. We discuss our thermal management policy in detail in Section 3. Experimental results are reported in Section 4. Finally, we conclude the paper in Section 5.

2. SYSTEM INFRASTRUCTURE

A tile-based network-on-chip architecture [13] is targeted here. Each tile is a processor with dedicated memory and an embedded router. It will also be referred to as processing element (PE) in this paper. All the processors and routers are connected by an on-chip network where information is communicated via packet transmission. Although it has been pointed out by [12] that the thermal impact of the on-chip network is not negligible, in this paper, we assume that there is very limited inter-processor communication and we focus only on the thermal issues related to the processors. We refer to the cores that can reach to each other via one-hop communication as the *nearest neighbors*. The proposed DTB algorithm migrates tasks among nearest neighbors in order to reduce overhead and minimize the impact on the communication bandwidth. We assume an existence of temperature sensor on each core. A temperature sensor can be a simple diode with reasonably fast and accurate response [5].

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. The proposed DTB algorithm is implemented as part of the OS based resource management program which performs thermal-aware task scheduling and 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 OS. Each task occupies a slice of operating time. The OS switches from one task to another when the time slice expires. The scheduling intervals of different cores do not have to be synchronized.

3. DISTRIBUTED THERMAL BALANCING POLICY

In this section we present the details of the distributed proactive thermal balancing (DTB) policy. Table 1 summarizes the notations that will be used in this paper.

As we mentioned before, each PE_i is a preemptive system and has a set of tasks LT_i. Each task occupies an equal slice of execution time t_{slice}. Between two execution intervals is the scheduling interval. Our DTB policy is performed in scheduling interval. The PE also switches from one task to the next task at the scheduling interval.

Table 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

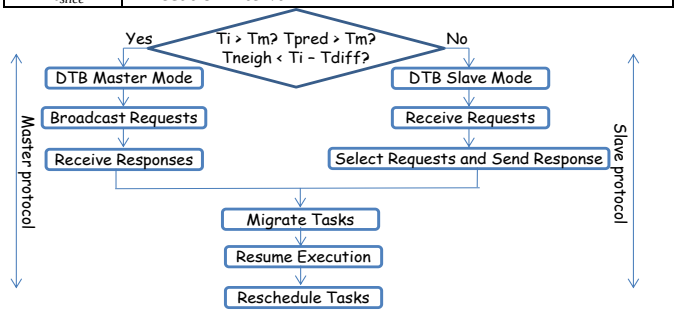

Figure 1. Master-Slave execution protocol

The DTB policy basically can be divided into 3 phases: temperature checking and prediction, information exchange and task migration. Figure 1 shows the flowchart of the DTB execution in the ith core. A DTB agent is initially neutral. It will

enter the master mode if any of the three scenarios are true: (1) the local temperature T_i reaches a threshold T_m (in this case, the DTB will first stall the processor to let it cool down before it enters the master mode), (2) the predicted future peak temperature exceeds the threshold T_m, (3) the temperature difference of the local core and the neighbor core exceeds the thermal balancing threshold T_{diff}. Otherwise, it will enter the slave mode. A master DTB agent issues a task migration request to its nearest neighbors which are DTB slaves.

Because the scheduling intervals in all processors are not synchronized, the request is not likely to be checked and responded by the DTB slaves right away. On the other hand, because all cores adopt the same execution and scheduling interval, it is guaranteed that all the DTB slaves will response within one t_{slice} after the request is issued. The asynchronous communication between master and slave DTB agents is explained by the example shown in Figure 2. It shows a single execution cycle of DTB policy starting from temperature check phase to task migration. When a master first enters its scheduling interval, it broadcasts a thermal balancing request in its neighborhood and then continues task execution.

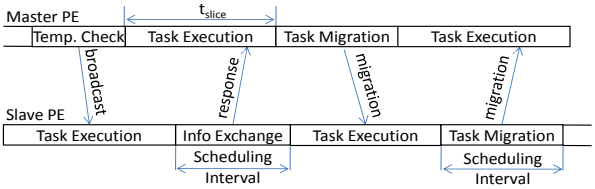

Figure 2. Master-slave communication

The slave will not respond until it reaches the next scheduling interval, when it selects a master from multiple requests and sends a response. In case of no requests, the PE resumes normal execution in next time slice. In case of multiple master requests, the slave selects a master which has the highest average power consumption. Response to this master PE includes details of slave's workload LT_s, its average steady state temperature $temp_s$, and PE id, etc. The slave is then locked to this master until it receives the migration request from the master or is released by the master.

After receiving the response, the master decides which tasks to migrate during its next scheduling interval and sends the migration command to slave. The tasks are migrated from master to slave at this time. After sending a response, the slave ignores any possible incoming request from other 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 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 master and slave if the workload difference between them exceeds the threshold nt_{diff}, this is measure by $\left||LT_i| - \left|LT_j\right|\right| > nt_{diff}, j \ni N_i$. The detailed workload balancing policy is presented in section 3.5. Secondly, if there is no workload imbalance, the DTB-M 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 balanced workload that produces fewer hot spots. The DTB-M policy can be divided into two parts. Both of the techniques have quadratic complexity to the number of tasks in the local task queue. The first technique is a steady state temperature based migration policy (SSTM). It considers the long term thermal behavior of tasks, and distributes tasks to cores based on their different heat dissipation ability. The second technique is a temperature prediction based migration policy (TPM), which predicts the peak temperatures of different task combinations when making 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 first technique considers long term average thermal effect and the second technique considers 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. Section 3.1 presents the temperature prediction model that will be used to trigger the master DTB and make the migration decision in TPM. Section 3.2 and 3.3 provide the details of the SSTM and the TPM policies and section 3.4 discusses how these two policies work together in DTB-M.

3.1 Temperature Prediction Model

For a multitasking system (e.g. Linux), each task in a task set could get a fair share of CPU time. Although the temperature change is fast and large when running these tasks, the peak temperature change is slow, as shown in [4]. Because the peak temperature is the main reason for thermal emergency, here we are interested in predicting the processor's peak temperature in the near future given the set of tasks on this processor.

The peak temperature of a processor is a function that depends not only on the power consumption of tasks running on it, but also on the temperature of its neighboring processors. It is difficult to determine the peak temperature analytically, therefore we employed a neural network [7] prediction model to find the relation between the peak temperature and related parameters. Our neural network predictor takes a set of inputs, and predicts the peak temperature in the near future. The inputs to the predictor include (1) the average power consumption of all tasks in the processor, because the average power consumption determines the average temperature level of the processor, (2) the power consumption of the task with largest average power, because a high power task is more likely to produce peak temperature (3) the recent highest and lowest temperatures of its neighbor processors in a history window.

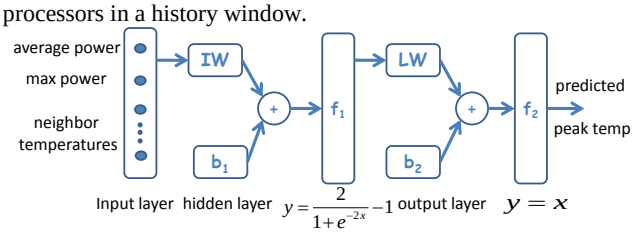

Figure 3. Neural network predictor architecture

Figure 3 shows the architecture of our neural network predictor. The predictor has one hidden layer and one output layer. There are $s = (2+2\times nn)$ input elements, where nn is the number of the nearest neighbors of a processor, m neurons in hidden layer and one neuron in output layer. We set m to be 7 to get a good tradeoff between prediction accuracy and computation amount. The IW, LW, b_1, b_2 are m by s input weight matrix, 1 by m layer weight matrix, 1 by m bias vector and 1 by 1 bias vector respectively. They will be trained during training process and fixed when we use the model to make prediction. The transfer functions for the hidden layer (f_1) and the output layer (f_2) are the *tansig* function and the *purelin* function respectively. Let P_0 denote the input

vector, the output of hidden layer can be calculated as P_1 = *tansig*($\boldsymbol{IW}\times P_0+b_1$), and the output for the output layer is P_2 = *purelin*($\boldsymbol{LW}\times P_1+b_2$), which is the predicted peak temperature.

The neural network predictor is trained using the fast and memory efficient Levenberg-Marquardt algorithm [7]. The training set is generated by running 500 groups of random picked synthetic workload on the many-core system and recording the peak temperature of each PE for different workloads. 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. It involves *ms+m* multiplications and *ms+2m+1* additions. Unlike other predictor models [4][11], we do not invoke the prediction at every time step. The predictor will be invoked when task in the PE changes, e.g. a new task has been migrated in, or when the core temperature exceeds the predicted value. This case could happen sometimes, for example, a PE made a prediction when its neighbors are cool. After its neighbors gradually heat up, the previous prediction will no longer be valid.

3.2 Steady State Temperature Based Task Migration (SSTM)

The SSTM policy balances high power tasks and low power tasks among neighbor PEs to optimize the average steady state temperature of the whole chip. It considers the lateral heat transfer between neighbor PEs and different heat dissipation capabilities of PEs. The SSTM policy assumes that the workload (i.e. the power consumption) of a PE is time invariant.

Let n denote the number of all thermal nodes in the system, including those in the heat sink layer and heat spread layer. 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. Let TSS and P denote vectors of TSS_i and P_i, $1\le i\le n$. 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

$$TSS = \boldsymbol{G}^{-1}P \qquad (1)$$

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

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

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 P_j of other nodes does not change. The coefficients g_{ij} and D_i $1\le i, j\le N$ can be obtained by offline analysis. Equation (2) shows that the steady state temperature of each 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 had some task exchanges, and their average power consumptions altered by ΔP_i and ΔP_j respectively. Using equation (2), the total steady state temperature change of all processors after task migration can be calculated as:

$$\sum_{k=1}^{N} \Delta T_k = G_i \cdot \Delta P_i + G_j \cdot \Delta P_j, \qquad (3)$$

where $G_i = \sum_{m=1}^{N} g_{mi}$, $G_j = \sum_{n=1}^{N} g_{nj}$. As we mentioned earlier, the goal of the SSTM policy is to reduce the average steady state temperature of the many-core system. In another word, it exchanges task pair to keep $\sum_{k=1}^{N} T_k$ decreasing, i.e. $\sum_{k=1}^{N} \Delta T_k < 0$. In this way, the master can maintain fairness of workload and reduce its own operating temperature as well as the system's steady state temperature. Algorithm 1 gives the SSTM policy. A master DTB agent in PE_i first forms all task pairs (τ_i, τ_j), $\tau_i \in LT_i, \tau_j \in LT_j, j \in N_i$ and $P_{\tau_i} > P_{\tau_j}$. Then for each task pair, equation (3) 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.

Algorithm 1 SSTM

1. **for** each $\tau_i \in LT_i$
2. **for** each $\tau_j \in LT_j, s.t. j \in N_i, P_{T_i} > P_{T_j}$
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(τ_i, τ_j)
6. } **while** ($\Delta T_{min} < 0$)

3.3 Temperature Prediction Based Migration (TPM)

The SSTM reduces the average steady state temperatures of the whole chip. However, it does not consider the temporal variation of the workload (i.e. the power consumption) on each processor and hence it may not be able to reduce the local peak temperature. The workload/power consumption of a processor is constantly changing because each processor is shared by a set of tasks with different power and thermal profiles. In order to capture the local temperature variation, the prediction model introduced in section 3.1 is used to decide whether a migration is beneficial or not.

Algorithm 2 TPM (Slave Process)

1. $S = \Phi$;
2. **for** each task $\tau_i \in LT_i$, (LT_i is the list of tasks on master i)
3. **for** each task $\tau_j \in LT_j$, (LT_j is the list of tasks on slave j)
4. **If** (Predict_Thermal_Emergency(τ_i) = FALSE)
Insert (τ_i, τ_j) to S;
5. **If** ($S \neq \Phi$)
6. Return (τ_i, τ_j) to master, (τ_i, τ_j) $\in S$ and $P_{\tau_i} \le P_{\tau_j}$, $\forall(\tau_i, \tau_j) \in S$;
7. **Else** Return NULL to master;

Algorithm 2.1 TPM (Master Process)

1. Let S = {(τ_i, τ_j) | (τ_i, τ_j) is an offer from a slave}
2. Let T = {(τ_i, τ_j) | (τ_i, τ_j) ϵS and P_{τ_i} is the maximum in S }
3. Select a task pair (τ_i, τ_j) ϵT, *s.t.* P_{τ_j} is the minimum in T
4. Swap (τ_i, τ_j)

Algorithm 2 shows the main computation of the TPM policy which is performed by the slave DTB agent. For each task τ_i on the master PE, the slave DTB agent employs the prediction model to determine whether the local temperature will exceed the thermal threshold T_m in the near future after exchanging τ_i with a local task τ_j. Among those tasks that can safely be exchanged with a task running on master PE, the one that has the lowest power is selected and sent to the master as a potential offer for task migration/exchange. On the master side, algorithm 2.1 is executed. First, the master agent selects the one with the highest power among the tasks that have been offered for task exchange. If this task τ_i receives multiple offers, the master agent selects the offer that exchanges τ_i with a task τ_j that has the lowest average power.

3.4 Overall Task Migration and Scheduling

Our DTB-M policy consists of both SSTM and TPM. 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, while low power tasks can be moved to the PEs that are more easily to heat up. On the other hand, the TPM algorithm mitigates the local hot spots and balances the thermal gradient.

After a master DTB 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 task. When the slave receives the request, it performs the TPM algorithm. In the reply message, it sends the master the task pair that is found by the TPM algorithm and also the list of its local task. The master then performs SSTM. If SSTM algorithm found task pair whose exchange can reduce the average temperature of the whole chip, then the master will issue a task migration command. If the SSTM algorithm cannot find any task pair for exchange, the master DTB performs the TPM algorithm.

After task migration finishes, we employ a simple technique to schedule the execution of tasks based on their average power consumption for both master and slave PEs. All tasks in a PE's run queue are sorted according to descending order of 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 start a new round of scheduling again. It is a simple yet effective scheduling technique that tries to reduce the operating temperature of the core.

3.5 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}, this is $\left||LT_i| - |LT_j|\right| > nt_{diff}, j \ni N_i$. The master will pick the slave which gives the maximum workload difference. The workload balancing policy is based on SSTM described in 3.2. Firstly, G_i and G_j are evaluated (see equation (3)). Then, tasks are migrated from the PE with more tasks to the PE with fewer tasks one by one until the task difference is less than or equal to one. In every migration, (3) is computed and the task which minimize the $\sum_{k=1}^{N} \Delta T_k$ will be selected. It can be proved that if $G_i > G_j$ and $|LT_i| > |LT_j|$, the migration from PE_i to PE_j will start from the highest power task in PE_i. On the other hand, if $G_i > G_j$ and $|LT_i| < |LT_j|$, the migration from PE_j to PE_i will start from the lowest power task in PE_j.

4. EXPERIMENTAL RESULTS

We implemented an event driven behavioral simulator of a multicore system using C++. Hotspot [9] is integrated to our multicore simulator to simulate the system thermal behavior. Though 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 evaluated the proposed thermal management policy using both static workload and dynamic workload. The system performance is characterized by the number of completed jobs within a given period of time. We assume that the temperature threshold to trigger thermal throttling is 78 °C and during thermal throttling, the CPU stalls its current execution. However, other low power techniques such as DVFS can easily be integrated into this framework. In all experiments, we set nt_{diff} = 2, t_{slice} = 100ms, T_m = 80 °C, T_{diff} = 10 °C for the DTB policy.

We compared our migration policy with the state-of-the-art Predictive Dynamic Thermal management (PDTM) policy proposed in [11]. PDTM predicts a running process's temperature based on its temperature history using Recursive Least Square method and based on the process's steady state temperature. The PDTM policy moves a process to a core which is predicted to be cold in the near future before the process reaches high temperature on its current running core. Therefore the system temperature can be reduced. The proposed DTB policy and the reference policy are compared from the following perspectives.

- *Hotspot*: The time spend above a temperature threshold which is 80 °C in our case.
- *Grad*: i.e. thermal gradient, the percentage of time that any two cores have more than 15°C temperature difference.
- *NT*: The number of tasks completed within a given period of time. This is our representation of performance in a system.
- *Mig:* total number of migrations occurred during execution. This is our representation of overhead.

We carried out experiments using power sequences collected from real applications. We used 9 different CPU benchmarks comprising of 3 SPEC2K benchmarks (bzip2, applu, mesa), 4 Mediabench applications (mpeg2enc, mpeg2dec, jpegdec, jpegenc) and 2 telecom applications (crc32 and fft) from MiBench benchmark suite. We collected cycle level power trace by modifying the Wattch power analysis tool [3]. The average power consumptions and steady state temperatures of each task are summarized in table 2. 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 2 Average power and steady state temperature of CPU benchmarks

Bench marks	crc32	mp2 enc	mp2 dec	fft	applu	mesa	bzip2	jpeg dec	jpeg enc
Avg. Power (mW)	24.4	19.4	19	18.5	17.4	17.3	13.3	10.7	10.4
Steady Temp. (°C)	99.42	84.17	82.95	81.42	78.07	77.76	65.56	57.63	56.72

4.1 Performance with Static Workload

In the first experiment, we apply the DTB-M policy in a system with statistic workload. The workload consists of 144 tasks. Each task is randomly selected from the 9 benchmarks mentioned above. We determine the selection probability of a benchmark based on its average power consumption so that the average power consumption of the 144 tasks can follow a desired distribution. Five power consumption distributions are tested in this experiment.

Uniform distribution evenly generates tasks with different 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. Inverse normal distribution generates more high power tasks and low power tasks than the medium power tasks.

Table 3 shows the comparison between the DTB policy and the reference policy over those 5 different distributions. We can see that DTB reduces the thermal gradients by 66.23% and hotspots

by 66.79% and improves the performance by 40.21% while maintaining a 33.84% lower migration overhead. The reason that the DTB gives improved performance is because the DTB exchanges tasks and tends to distribute workload evenly across the system and hence increases the parallelism while PDTM will move tasks from different hot PEs to the same cooler PE, thus making the load on different PEs unbalanced. PDTM also creates much larger thermal gradients for the same reason. Without load balancing, a PE may become idle and hence drop to a very low temperature before new tasks moved in.

Table 3 Performance with static workload

Workload distribution	Policy	Uni.	Tri. (cool)	Tri. (hot)	Norm.	Inv. Norm.
NT	**DTB-M**	144	144	144	144	144
	PDTM	84.4	90.8	82.9	86.2	86.2
	%Impr.	41.39	36.94	42.43	40.14	40.14
Mig. #	**DTB-M**	489.6	273.8	566.6	468.2	459.6
	PDTM	582	418	1392.9	1009.3	488.5
	%Impr.	15.88	34.50	59.32	53.61	5.92
Grad.	**DTB-M**	3.66%	2.62%	4.02%	2.95%	3.76%
	PDTM	5.73%	9.02%	14.37%	9.02%	10.83%
	%Impr.	36.19	70.89	72.00	67.28	65.31
Hotspot	**DTB-M**	115	135	147	238	150
	PDTM	354	300	991	834	332
	%Impr.	67.51	55.00	85.17	71.46	54.82

Note that although each time the DTB migrates two tasks (one task in and one task out), the total number of migrated tasks of DTB policy is still much less than that of PDTM. This is because the DTB will analysis migration decisions first. If it finds that a migration could cause hot spots on target PE, it will stop the migration, therefore reduces the number of unnecessary migrations. DTB also maintains a good mixture of high power and low power tasks in each PE, the low power tasks can help reduce the heat generated by high power tasks. Thus the high power tasks do not have to be moved among neighbor PEs again and again. This mixture of high power and lower power tasks also reduces the hotspots. On the other hand, PDTM does not analyze the thermal effect a migration would bring to the target PE. Therefore, a high power task has a high chance to be moved among different PEs frequently. We also observed in our experiment that the average distance of each migration in PDTM is 3.41 hops while the distance in DTB is always 1 hop. Our experimental results also show that the neural network predictor is very accurate. The average difference between the predicted peak temperature and the actual peak temperature is only 1.5 ^{o}C. And the percentage of times the actual temperature of a PE exceeds T_m, but our predictor failed to catch that is within 6%.

Table 4 Performance with dynamic workload

Policy	NT	Mig. #	Grad.(%)	Hotspot
DTB-M	161	965.7	5.82%	174
PDTM	125.8	1279.4	13.11%	493.6
Improvement (%)	27.98	24.52	55.62	64.75

4.2 Performance with Dynamic Workload

In the second experiment, we further introduce some randomness in the workload to model a dynamic system where new tasks enters and existing tasks leaves. The initial task set consists of 144 tasks generated as described in section 4.1 with uniformly distributed power consumptions. Instead of fixed durations, the execution time of a task is uniformly distributed between 15 to 30 time slices, which is equivalent to 1.5 to 3 seconds. Every execution interval, a new task is generated on a PE with 0.02 probability. The execution time of the new tasks follows the same distribution. Table 4 gives the performance comparison between DTB-M and PDTM. The experimental results show that DTB-M can effectively improve the system performance and balance the temperature in dynamic workload as well.

5. CONCLUSION

In this paper, we proposed a distributed thermal balancing policy which stabilizes the operating temperature and improves the performance of many core systems. A lightweight agent is proposed for each core which independently monitors the temperature and work towards maintaining a feasible thermal profile and improve the core performance. Together all cores collaboratively balance the thermal gradient on chip. Experimental results show that performance is improved with reduced overhead.

6. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0845947

7. REFERENCES

[1] S. Borkar, "Thousand Core Chips – A Technology Perspective," *In Proc. Design Automation Conference*, June 2007.

[2] D. Brooks and M. Martonosi, "Dynamic Thermal Management for High Performance Microprocessors," *In Proc. Int. Symp. High Performance Computer Architecture*, pages 171-182, Jan. 2001.

[3] D. Brooks, V. Tiwari and M. Martonosi, "Wattch: A Framework for Architectural Level Power Analysis and Optimizations," *In Proc. Int. Symp. Computer Architecture*, pages 83-94, June 2000.

[4] A. Coskun, T. Rosing and K. Gross, "Proactive Temperature Management in MPSoCs," *In Proc. Int. Symp. on Low Power Electronics and Design*, pages 165-170, Aug. 2008.

[5] J. Donald and M. Martonosi, "Techniques for Multicore Thermal Management: Classification and New Exploration," *In Proc. Int. Symp. Computer Architecture*, pages 78-88, June 2006.

[6] T. Ebi, M. Faruque and J. Henekl, "TAPE: Thermal-Aware Agent-Based Power Economy for Muti/Many-Core Architectures", *In Proc. Int. Conf. on Computer-Aided Design*, pages 302-309, Nov. 2009.

[7] R. Jayaseelan, T. Mitra, "Dynamic Thermal Management via Architectural Adaption", *In Proc. Design Automation Conference*, pages 484-489, Jul. 2009.

[8] F. Mulas, M. Pittau, M. Buttu, S. Carta, A. Acquaviva, L. Benini, D. Atienza and G. De Micheli, "Thermal Balancing Policy for Streaming Computing on Multiprocessor Architectures," *In Proc. Design Automation and Test in Europe*, pages 734-739, March 2008.

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

[10] 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 Proc. Int. Solid-State Circuits Conf.*, pages 98-589, Feb. 2007.

[11] I. Yeo, C. Liu and E. Kim "Predictive Dynamic Thermal Management for Multicore Systems," *In Proc. Design Automation Conf.*, pages 734-739, June 2008.

[12] L. Shang, L. Peh, A. Kumar and N. Jha, "Thermal Modeling, Characterization and Management of On-chip Networks," *In Proc. Int. Symp. Microarchitecture*, Dec., 2004.

[13] W. Dally, B. Towles "Route packets, not wires: on-chip interconnection networks," *In Proc. Design Automation Conf.*, Jun. 2001.

[14] S. Liu, J. zhang, Q. Wu and Q. Qiu, "Thermal-Aware Job Allocation and Scheduling for Three Dimensional Chip Multiprocessor," *in Proc. International Symposium on Quality Electronic Design*, Mar. 2010.