A Multi-Agent Based Thermal Aware Task Migration Scheme in Multi-Core System

ISSN 2319-9725

Kashif Ahmed
M.tech (VLSI and ES), 2nd Year,
Dept. of ECE
CMR Institute of Technology, Bangalore, India

Dr. G. Indumathi
HOD, Dept. of ECE
CMR Institute of Technology

Abstract: As feature sizes decrease, power dissipation and heat generation density exponentially increase. Thus, thermal hot spots and large temperature gradients in Multiprocessor Systems on Chip (MPSoCs) can seriously impact the system performance, reliability, cost, and leakage power. Increasing complexity of the system makes it more difficult to perform thermal management in a centralized manner. In this paper, a framework is proposed for distributed thermal management in many-core systems where balanced thermal profile can be achieved by proactive task migration among neighboring cores. The framework has a low cost agent residing in each core that observes the local workload and temperature and communicates with its nearest neighbor for task migration and exchange. By choosing only those migration requests that will result in balanced workload without generating thermal emergency, the proposed framework maintains workload balance across the system and avoids unnecessary migration. Compared with existing proactive task migration technique, the approach used in this paper generates less hotspot, less migration overhead with negligible performance overhead.

Index Terms: Distributed thermal balancing migration (DTB-M), neural network model, master-slave communication, processing element (PE), task migration.
1. Introduction:

With remarkable increase in number of transistors integrated on a single chip, the multi-core technology may evolve to thousands of core era. TILE64 by Tilera consisting of 64 cores and 80-tile NoC by Intel are few examples of such systems. Though the performance delivered by the multi-core technology is incredible, they have to face substantial power and thermal challenges.

Exponential increase in the power density uplifts the peak temperatures of chip and unbalances the thermal gradient. This reduces the lifetime of chip, deteriorates its performance, affects its reliability and increases the cooling cost.

In a mapped multi-core system, varied workload of applications leads to imbalance in power and temperature among different cores. Such spatial temperature variation, referred to as thermal gradient, creates hotspot on the chip. Hotspot increases the clock skews and decreases performance and reliability.

Different dynamic thermal management (DTM) techniques have been proposed for many-core systems, such as dynamic voltage and frequency scaling (DVFS), thread management, and clock gating. These techniques ensures that the system run under a safe fixed temperature constraint. But many of these techniques have centralized approach i.e., they require a controller to monitor the temperature and workload distribution of each core on the entire chip and make global resource allocation decisions. The drawback of such centralized approach is that they do not have good scalability.

In this paper, a distributed thermal management framework is proposed to achieve thermal balance by proactive thermal throttling as well as thermal aware task migrations among neighboring cores. There exists a low cost agent in each processing element (PE), which observes the workload and temperature of PE while simultaneously exchanging the tasks with its nearest neighbor. The goal of the proposed task migration is to match PE’s heat removal capability to its workload and at same time create proper mix of hot and cool tasks running on it. Compared with centralized approach, the proposed framework achieves much better scalability, as each agent monitors only local PE and communicates with its nearest neighbor. This technique is referred to as distributed thermal balancing migration (DTB-M).
Two proactive migration schemes have been proposed in this paper. A steady-state temperature-based migration (SSTM) and temperature prediction-based migration (TPM). SSTM considers long term thermal behavior of tasks, and distributes tasks to PE’s based on different heat removal capabilities. TPM predicts the thermal impact of different workload combinations and adjusts the task allocation in a neighborhood so that all the PE’s gets a good mixture of hot tasks and cool tasks. The two methods together provide accelerating improvements that reduces thermal gradients and prevents thermal throttling events.

A neural network based temperature predictor is proposed in this paper. Based on the workload of a PE, it predicts the future peak temperature and some preliminary information from neighboring PE’s. Compared with other thermal predictors, the neural network predictor has very low computation complexity and it can give accurate prediction right after task migration.

2. Related Work:
Modern day microprocessors handle thermal emergencies through various DTM mechanisms. Techniques at micro architecture level have been well explored in [2] and [3]. At system level, voltage/frequency scaling, task scheduling, task allocation, and thread migration can be combined to leverage the temperature reduction on MPSoCs. In [4], frequency assignment has been formulated as a convex optimization problem and optimum solutions can be solved offline. Online voltage/frequency scaling techniques often utilize a feedback controller to adjust voltage/frequency settings. The authors of [5] use a linear quadratic regulator to adjust the frequency assignment online for thermal balancing. In [6], chip power consumption and operating temperature are controlled to a desired point by a multiple-input and multiple-output (MIMO) controller based on the model predictive control theory. Coskun et al. [7] proposed a light weight probability based scheduling method which could achieve better temporal and spatial thermal profile.

In a many-core system, the heat dissipation capability differs from processor to processor. In [8], an algorithm is proposed to map and schedule tasks based on the thermal conductivity of different processors. In [9], Sharifi et al. proposed a task allocation and frequency assignment algorithm which use exhaustive search to find a location and a voltage/frequency setting for incoming tasks to achieve energy saving and balanced temperature. Michaud et al. [10]
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 [11], and they derived an approximate analytic expression of system throughput depending on several parameters of interest.

Thermal management of on-chip interconnect network is addressed in [12]. Shang et al. first proposed an architecture thermal model for on-chip networks. Based on this model, they further proposed ThermalHerd, a framework which uses distributed thermal throttling and thermal aware routing to tackle thermal emergencies.

Proactive thermal management based on runtime task migration has been proposed in references [13] and [14]. Both of them predict the future temperature as a projection of the history temperature trace. Although these predictive models are very accurate in most circumstances, they have some limitations. First of all, both models have to be updated and adjusted at runtime. This could introduce adaption overhead. Second, both models predict the future temperature solely from the temperature history. For a system with frequent task migrations, history trace does not reflect future temperature because the workload changes dramatically. The predictor cannot give accurate prediction until it has adapted to the new workload which may take a long time.

Unlike the prediction model proposed in [13] and [14], the neural network-based prediction model proposed in this paper can overcome the limitations mentioned previously. The 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.

3. Distributed Thermal Balancing Policy:

Table I gives the notations that will be used in this paper. Each processing element PE_i is a preemptive system and contains set of tasks LT_j. Each task occupies an equal execution time slice t_{slice}. There exists a scheduling interval between two execution intervals. Thermal balancing and switching of tasks by PE’s is performed in the scheduling interval.
Table 1: List of Symbols and Their Definitions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$LT_i$</td>
<td>The list of tasks running on core $i$</td>
</tr>
<tr>
<td>$</td>
<td>LT_i</td>
</tr>
<tr>
<td>$T_i$</td>
<td>A task in $LT_i$</td>
</tr>
<tr>
<td>$P_i$</td>
<td>The power of $T_i$</td>
</tr>
<tr>
<td>$T_i$</td>
<td>Current temperature of core $i$</td>
</tr>
<tr>
<td>$N_i$</td>
<td>The set of nearest neighbors of core $i$</td>
</tr>
<tr>
<td>$T_m$</td>
<td>Temperature threshold to trigger the DTB-M algorithm</td>
</tr>
<tr>
<td>$T_{th}$</td>
<td>Threshold to trigger thermal balancing</td>
</tr>
<tr>
<td>$w_{max}$</td>
<td>Threshold to trigger workload balancing</td>
</tr>
<tr>
<td>$t_{idle}$</td>
<td>Execution interval</td>
</tr>
</tbody>
</table>

Power consumption of all tasks in $LT_i$ is referred to as workload of $PE_i$, and the different combination of tasks in $LT_i$ is referred to as workload patterns of $PE_i$. Operating System (OS) easily observers both the information.

Basically DTB-M policy is divided into three phases: temperature checking and prediction, information exchange and task migration. Flowchart of DTB-M execution in $i$th core is shown in fig. 1.

Figure 1: Master-Slave execution protocol

A DTB-M master initiates a task migration request while the DTB-M slave responds to it. A DTB-M agent can either be in master or slave mode. However by default the agent will be in slave mode. For the DTB-M agent to enter the master mode, the following three scenarios must be true:

i. The local temperature $T_i$ reaches a threshold $T_m$ in the last execution interval, resulting in hotspot generation. The DTB-M agent throttles the processor to let it cool down and then continues the execution.

ii. The predicted future peak temperature exceeds $T_m$ and current peak temperature is larger than $T_m - \delta$, where $\delta$ is temperature margin.
iii. The temperature difference between the local core and neighbor core exceeds the thermal balancing threshold $T_{\text{diff}}$.

The first and second scenario indicates hotspots generation while the third scenario indicates high thermal gradients. Hence, a task migration request is initiated by the master to its nearest neighbor. Due to unsynchronized scheduling intervals in all the processors, the slave agents will not respond to the request immediately. They will respond within one $t_{\text{slice}}$ after the request is initiated.

Fig. 2 shows an example of asynchronous communication between the master and slave agent.

![Figure 2: Master-Slave communication](image)

When an agent enters its scheduling interval and becomes a master, it will broadcast a migration request to its nearest neighbor and then continues task execution. The slave agent will not respond to the request until it reaches its next scheduling interval. It then checks its message queue for incoming requests. If there are no requests, slave PE continues normal execution in its next $t_{\text{slice}}$. In case of multiple master requests, the slave selects a master having highest average power consumption. Response of slave PE to master PE includes its ID, details of slave’s workload and its recent operating temperature. The slave then locks itself to this master until the master releases it.

Once the master PE receives the response it will decide which task has to be migrated in its next scheduling interval and sends the migration command to the slave PE. During this time tasks are migrated from master to slave. On the other hand the slave, after sending a response, ignores any possible incoming requests from other master agents until it receives the migration command from the original master. The DTB-M policy cycle ends by migrating tasks from slave to master during this time.
The master agent considers load balancing as well as thermal balancing to make migration decisions. First, load balancing process is triggered which migrates tasks one way to balance the workload between master and slave if workload difference between them exceeds the threshold $n_{\text{diff}}$, which is measured by $|LT_i| - |LT_j| > n_{\text{diff}}$, $j \in N_i$. If workload is balanced then thermal balancing process is triggered.

The idea behind DTB-M policy is to exchange tasks between neighboring PE’s, so that each PE can get a set of tasks that produces fewer hotspots. DTB-M policy consists of two techniques: SSTM and TPM. Both techniques are complementary to each other with SSTM focusing on long term average thermal effect and TPM focusing on short term temporal variations. 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.

4. **Neural Network Based Temperature Prediction Model:**

A neural network model is composed of number of interconnected neurons, referred to as PE’s, working together to solve a specific problem. The model can be trained through a standard learning process after which it is used to provide projections on the new data.

![Two layer neural network predictor architecture](image)

Fig. 3 shows a two layer neural network model for prediction of peak temperature. It consists of two layers, hidden layer ($f_1$) and an output layer ($f_2$). They are defined by the two equations:
\[
tanh(x) = \frac{2}{1 + e^{-2x}} - 1
\]
\[
purelin(x) = x.
\]

Input to neural network is selected from set of features relevant to peak temperature prediction. These features are divided into two categories: features collected from local processor and features collected from neighbor processors. The local feature consists of two variables that give average and maximum power consumption of the tasks running on the local processor. For \(i\)th core, they are calculated as \(\sum_{\tau_i \in LT_i} P_{\tau_i} / |LT_i|\), and \(\max_{\tau_i \in LT_i}(P_{\tau_i})\). The neighbor feature consists of three variables for each neighboring processor, which specifies average and maximum power consumption of each neighboring processor along with the recent highest temperature in a history window.

The training process of the neural network based peak temperature predictor uses fast and memory efficient Levenberg-Marquardt algorithm [15] provided by Matlab neural network toolbox. Since neural network model is trained for each core on the chip separately, these models are able to capture the core to core process variations.

Accuracy of the predictor can be improved by adding more neurons in the hidden layer. The relationship between the size of neural network and prediction accuracy is shown in table II.

<table>
<thead>
<tr>
<th>Order</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>7</th>
<th>10</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSE</td>
<td>0.068</td>
<td>0.225</td>
<td>0.440</td>
<td>0.038</td>
<td>0.099</td>
<td>0.044</td>
<td>0.045</td>
<td>0.048</td>
</tr>
<tr>
<td>Avg. err</td>
<td>2e-05</td>
<td>-0.0020</td>
<td>-0.0017</td>
<td>-0.0013</td>
<td>0.0077</td>
<td>-0.0030</td>
<td>-0.0018</td>
<td>-0.0010</td>
</tr>
</tbody>
</table>

Table 2: Prediction Accuracy V/S Size Of Neural Network

The number of neurons in the hidden layer is specified by the first row and second row gives the Mean Square Error (MSE) of the estimation. As seen from the table, increasing the number neuron in the hidden layer will not increase accuracy but introduces higher computation complexity. Hence, only 1 neuron is used in all PE’s.

Unlike other prediction models [13] and [14], the neural network predictor was not invoked at every time step. Instead, it was invoked when the core temperature exceeds predicted value or when the workload pattern in PE changes. The workload pattern changes whenever a task is generated, completed or migrated. If a task has several phases, with each phase having different power and thermal characteristics, then each phase is considered as a single task. New prediction is then made whenever a phase change is detected.
For example, fig. 4 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 a many-core simulator. During 12 s simulation time, tasks migrate, start, or complete randomly. The workload information at different time is denoted at the bottom of the figure. While the blue line gives the trace of the real temperature, the green line gives the predicted peak temperature. As we can see, the predictor is invoked every time the workload pattern changes and it is able to track the peak temperature accurately.

![Figure 4: Temperature prediction of neural network model](Image)

5. Distributed Task Migration:

A. SSTM:

The SSTM policy balances high power tasks and low power tasks among neighbor PE have to lower the average steady state temperature of the whole chip. It considers the lateral heat transfer between neighbor PE’s and their different heat dissipation capabilities.

The main computation of SSTM is done in master PE. Algorithm 1 gives SSTM policy.

---

**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 ($\Delta T_{min} < 0$)
For example, assume PE_i and PE_j exchange some tasks, and their average power consumption altered by ΔP_i and ΔP_j respectively. A master DTB-M agent in PE_i first forms all task pairs (τ_i, τ_j), τ_i ∈ LT_i, τ_j ∈ LT_j, j ∈ N_i with Pτ_i > Pτ_j. For each task following equation is evaluated:

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

The task pair that gives 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.

**B. TPM:**

Even though SSTM policy is very effective, it has several limitations. First, SSTM may move all high power tasks in a neighborhood to one core whose thermal conductance matrix (G) value is minimum. Second, if G value of core is less than G value of all its neighbors, then the core may not be able to exchange its high power tasks with a low power task in its neighborhood when it is overheated, because this will increase the average steady state temperature of the chip.

To overcome the drawbacks of SSTM policy, TPM policy is utilized. 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.

Main computation of TPM policy is performed by slave. Algorithm 2 for slave and master process is as shown.

---

**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 τ_i ∈ LT_j
5. For each task τ_j ∈ LT_j
6. If (Pτ_i < Pτ_j)
7. \( T_p = \text{Predicted local peak temperature after task exchange} \)
8. If (\( T_p < T_m \)) return to the master and exit;
9. Return NULL to the master and exit;
The Algorithm 2 TPM (Slave Process) 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 \( \tau_i, \tau_j \) where \( \tau_i \in LT_i \) 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 \( \tau_i \) and \( \tau_j \) 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 TPM (Master Process) 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.

**C. Combined Migration Policy:**

The DTB-M policy proposed in this paper is a combination of both SSTM and TPM policies. When a 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).

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

**D. Workload Balancing Policy:**

Workload balancing is triggered when a master PE\(_i\) finds the workload difference between itself and a slave PE\(_j\) exceeds the threshold \(nT_{\text{diff}}\), that is \(||LT_i| - |LT_j|| > nT_{\text{diff}}\), \(j \in N_i\). 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, the following equation is evaluated:

\[
\sum_{k=1}^{N} \Delta T_k = G_i \cdot \Delta P_k + G_j \cdot \Delta P_j
\]

and the task which minimizes the \(\sum_{k=1}^{N} \Delta T_k\) will be selected. It can be proved that if \(G_i > G_j\) and \(|LT_i| > |LT_j|\), the migration from PE\(_i\) to PE\(_j\) will start from the task with the highest power. On the other hand, if \(G_i > G_j\) and \(|LT_i| < |LT_j|\), the migration from PE\(_j\) to PE\(_i\) will start from the task with the lowest power.

**6. Experimental Results:**

In this experiment a 9-core system with 3 x 3 grid is chosen, although the model can be scaled for any number of cores. It was assumed that the temperature threshold to trigger thermal throttling is 90°C and during thermal throttling, the CPU stalls its current execution. ModelSim SE Plus 6.2c and Xilinx ISE 12.1 tools have been utilized for simulation and synthesis. The codes are written in VerilogHDL and C (for neural network predictor model).
Fig. 5 clearly shows the migration of task from core1 to core4 and core 5 to core 6 when the task running on core1 and core 5 exceeds the temperature threshold. As per [1], the DTB-M policy reduces 29.8% hotspots and 80.68% migration overhead with only 0.98% performance overhead compared to the PTB thermal management.

7. Conclusion:

In this paper, a distributed thermal management framework was proposed for many-core systems. The framework resides an agent in each core to monitor the core temperature. Also the agent communicates and negotiates with neighboring agents to migrate and distribute the tasks uniformly over the entire system. The agent uses DTB-M policy for task migration. DTB-M policy is a combination of SSTM and TPM migration policies. SSTM policy distributes different tasks in a neighborhood based on their heat dissipation ability. TPM policy ensures a good mixture of hot and cool tasks on processors in a neighborhood.

A neural network based thermal prediction model was proposed in this paper. The model not only predicts the future temperature, but also helps the agents to evaluate the rewards of proposed migration offers.
References:


