Decentralized Thermal-aware Task Scheduling for Large-scale Many-core Systems

Yingnan Cui, Wei Zhang, Vivek Chaturvedi, Bingsheng He

***Abstract*—Technology scaling has enabled fast increase in the number of cores integrated in many-core systems. However, feature size shrinking also makes large-scale many-core systems vulnerable to thermal failures. Thermal-aware task scheduling is an efficient technique to reduce the run-time temperatures of many-core processors. Most existing thermal-aware task scheduling algorithms leverage centralized scheduling schemes to gather the overall information and generate the task schedule at a center scheduler. Although that scheme can achieve the optimal temperature reduction, however, it faces severe computation bottleneck and communication congestion when the many-core processors evolve to large-scale with hundreds or thousands of cores. In this paper, we propose a decentralized thermal-aware scheduling algorithm to address this problem in large-scale systems. Experiment results on various benchmarks show that our decentralized algorithm achieves significant improvement on scalability (up to 84.3% reduction in monitoring traffic) and similar benefits on temperature reduction (by 5%) when compared to the state-of-the-art thermal-aware scheduling algorithm.**

I. Introduction

Technology scaling has enabled a trend in which the number of cores integrated inside one chip grows rapidly [1]. According to the technology road map, thousand-core processors may come into reality in the near future [2]. However, as a negative side effect, technology scaling makes the many-core systems vulnerable to thermal malfunctions due to the increase in power density. Previous studies have shown that the average power density of 65nm IC is around $2W/mm^2$ [3], which could result in a chip temperature of above $90°C$. As the temperature of the chips rises, thermal-related malfunction phenomenon become more active and can damage the reliability and shorten the lifespan of the chips [4]. Examples of thermal malfunctions include negative biased temperature instability (NBTI), gate oxide break down and electromigration. All those malfunctions pose significant challenges to processor reliability. The shrinking of feature size also brings the problem of dark silicon [5], which severely limits the number of activated components in a chip. As pointed out by [6], [7], to solve dark silicon problem, traditional solution of using the *thermal design power* (TDP) as the power constraint is not sufficient. A more promising solution to the dark silicon problem is to apply thermal constraints to each of the cores in the processors. In conclusion, efficient thermal management techniques are critically demanded by modern processors.

In previous studies [8]–[12], *thermal-aware scheduling* has been adopted to solve the thermal problems for multi/many-core systems. Studies like [11], [12] adopt formal optimization techniques like linear integer programming to solve the thermal-aware scheduling problem in design time. On the contrary, studies like [8]–[10] use heuristic-based solutions to improve the efficiency of thermal-aware scheduling algorithms for run-time use. To facilitate the design of thermal-aware scheduling algorithms, efficient thermal estimation techniques are also proposed [13].

All the above mentioned studies adopt a centralized scheduler to manage the temperature of all cores in a many-core system. However, large-scale multi-core systems pose several challenges for centralized thermal-aware scheduling algorithms. Firstly, as the number of cores contained by a processor grows, the complexity and communication overhead of centralized scheduling algorithms also increases rapidly. For thousand-core processors, the overhead of scheduling algorithm could become a bottleneck for the performance and power of the system. Secondly, many large-scale multi-core processors adopt the voltage/frequency island (VFI) architecture to solve the power supply and synchronization problem brought by the large-scale integration [14]. In such processors, the chip is divided into several VFIs, each of which works on an individual voltage and frequency level regardless of others. The VFI architecture further increases the complexity of the centralized scheduling algorithms if the algorithms were to consider the communication between cores.

To address the above challenges, decentralized scheduling algorithm is a promising solution. Proposed in previous studies like [15], [16], the decentralized scheduling algorithms usually divide the processor into *clusters* and perform the scheduling or mapping within each cluster. A verification solution for decentralized scheduling algorithms is further proposed by [17]. The benefits of decentralized scheduling algorithms are as follows. Firstly, by limiting the number of cores considered in the scheduling, the complexity and communication overhead of the scheduling algorithms could be efficiently reduced. Secondly, by making each cluster cover the same range of a VFI, the architecture model to be considered by the scheduling algorithm is also largely simplified.

However, the previous decentralized approaches have been targeting performance or power optimization. In this paper, we propose a novel decentralized thermal-aware scheduling algorithm for large-scale many-core systems to minimize the temperature of the processors in the NoC system while meeting timing constraints of the applications. In the algorithm, the scheduling of applications is performed in a hierarchical manner. First, the resource demand of the application is summarized and a light-weight global agent selects a suitable cluster to assign the application considering its performance and temperature metrics. Then the cluster agent of the selected cluster performs the detailed scheduling of the application inside the cluster. In our design, a thorough study has been conducted to

achieve the best balance of the workload between the global agent and the cluster agents to minimize the monitoring traffic while retaining the necessary information communication to optimize the temperature reduction of the processors. We also propose an efficient cluster size adjustment method to dynamically merge the clusters to better fit the task graph considering the critical impact of the cluster size to the scheduling results. Experimental results show that our algorithm has significantly improved the scalability as the size of the many-core systems increases and a total amount of monitoring traffic up to 84.3% is reduced compared to a state-of-the-art centralized thermal-aware scheduling algorithm [11]. At the same time, the achieved average temperature is comparable to the centralized algorithm (5% higher) while all the timing constraints are satisfied.

The following sections of this paper are organized as follows. Section II introduces the preliminaries for our scheduling algorithm. Section III introduces the five-step scheduling flow of the decentralized thermal-aware scheduling algorithm. Section IV describes the detailed algorithms for each step in the scheduling flow. Section V presents the performance evaluation of the proposed algorithm through experimental results. Section VI summarizes the related work of thermal-aware scheduling algorithms. Section VII concludes the paper.

II. Preliminaries

In this section, we introduce the preliminary assumptions for the paper.

A. VFI-based Large-scale Multi-core Processors

Fig. 1 shows the architecture of the VFI-based many-core parocessor. The processor is divided into a number of VFIs, each of which has its individual power supply and clock circuits. Each VFI contains a number of cores which run at the same voltage and frequency level. In this study, we assume that there are two available frequency levels for each VFI. The higher frequency level, denoted as f_H, is used for normal execution of the applications. The lower frequency level, denoted as f_L, is used for low power execution to deal with thermal emergencies. In our study, the set of cores contained by the many-core system is denoted by M, and the total number of the cores in the system is assumed to be$|M|$. The i$^{\text{th}}$ core in the system is denoted by m_i, where $m_i \in M$. We assume that all the cores in the system are homogeneous.

The communication framework for the many-core system is network-on-chip (NoC), which is the mainstream choice for large-scale multi-core processors [18] due to its high scalability. Some experimental many-core systems also adopt NoC as the communication scheme [19]. In VFI-based processors, the communication channels that lie on the boarder of two VFIs have a synchronization unit to enable the communication between cores running at different frequency levels [14].

We use the model proposed in [14] to compute the communication delay and power consumption for VFI-based multi-core processors. The communication delay, denoted by t_{comm}, is computed by Eq. (1). In the equation, vol is the volume of the data in the communication, W is the channel width,

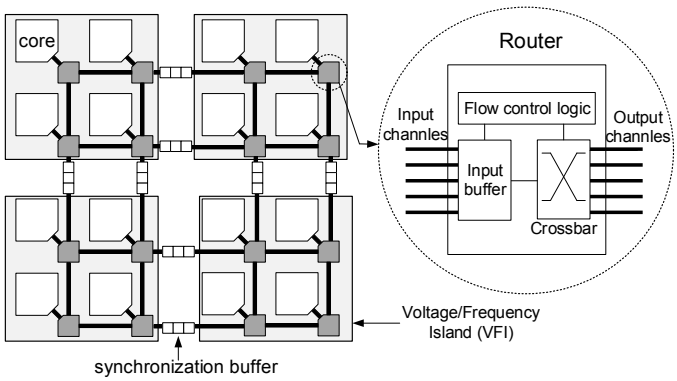


Fig. 1: A VFI-based multi-core system with 16 cores and 4 VFIs

μ_s is the number of cycles for the message to traverse a single router, P is the set of routers on the routing path of the message, and f_i is the frequency level of the i$^{\text{th}}$ router on the routing path, t_{sync} is the unit latency for synchronization buffers.

$$t_{comm} = \sum_{i \in P} \frac{\mu_s}{f_i} + t_{sync} \lceil \frac{vol}{W} \rceil \tag{1}$$

The energy consumed by the NoC for transmitting one bit of data, denoted by E_{bit}, is computed by Eq. (2), where E_L, E_B, E_S represents the energy consumed by the link, buffer and switching device in a router with the normal voltage level V_{DD}, V_i denotes the voltage level of the i$^{\text{th}}$ router on the routing path.

$$E_{bit} = \sum_{i \in p} (E_L + E_B + E_S) \frac{V_i^2}{V_{DD}^2} \tag{2}$$

B. Application Model

In this study, the workload of the many-core systems are assumed to be applications that are modelled by task graphs. As pointed out by [20], large-scale many-core systems are designed to meet the increasing demand for high performance computing applications. Such applications usually have large amount of thread-level parallelism and can be modeled by task graphs. Example applications like robot control, sparse matrix solver and SPEC Fpppp benchmark [21].

A task graph, denoted by $G(V, E)$, is a directed acyclic graph (DAG), where the vertex set V represents the sub-tasks in the application, and the edge set E represents the inter-task communications (usually in the form of input/output dependency). The task graph has a soft deadline, denoted by t_d. Each task of the application, denoted by v_i, has two predefined properties: the length of the task, denoted by l_i, and the power of the task, denoted by p_i. Each edge of the application, denoted by $e_{i,j}$, represents a message transmitting from task v_i to v_j. The edge also implies the inter-dependence relationships between the tasks connected by it, where task v_j can only start to run after v_i is finished.

In this study, we assume that the workload for the many-core system is a set of previously known applications. The task graph of each application, including all the detailed information, could be acquired through off-line profiling [21]. For

high performance data centers and some embedded systems, such assumptions are reasonable and can be found in various previous studies regarding scheduling problems [11], [15].

C. Thermal Monitoring, Estimation and Emergencies

In this study we assume that for each core in the system there is a thermal sensor to monitor the temperature of the core during run-time. The development of thermal diode has make it possible to attach a thermal sensor to each core in a multi-core processor [22]. Such assumption has also been adopted by many previous studies [23], [24].

In our scheduling algorithm, we need to estimate the temperature of a core in the future to make thermal-aware scheduling choices. In this study, we adopt the thermal estimation model proposed in [13], which considers both the power generation of the core and the thermal interference between neighbor processors. Eq. (3) gives the thermal estimation model, where τ_{pred} is the predicated temperature, τ_{curr} is the current temperature, τ_i is the temperature of a neighbor processor, p is the power consumption of the core, and α and β are parameters depending on the floorplan of the processor. We note that the equation is used to predict the temperature of the core after a certain period of time, which is called *the thermal prediction step* in this work. To balance the accuracy and the complexity of the scheduling algorithm, we set the thermal prediction step as 4ms as in [24].

$$\tau_{pred} = \tau_{curr} + \sum_{i \in M_{nb}} \alpha_i(\tau_i - \tau_{curr}) + \beta p \qquad (3)$$

As we have mentioned in Section I, in order to prevent thermal failures, it is better to set a threshold temperature for each cores in a multi-core processor. When the temperature of a core rises above the threshold, denoted as τ_{th}, we call this event as a *thermal emergency*. In this study, we assume when thermal emergencies happen, the frequency level of the VFI which contains the overheated core is going to be scaled down to the lower level f_L. After the temperature of that core falls under τ_{th}, the voltage and frequency level is going to be switched back to the normal level f_H. This is one of the most commonly used techniques dealing with thermal emergencies in previous studies [25], [26].

III. Cluster-based Decentralized Scheduling Flow

Since DVFS is a performance throttling thermal management technique, in this study, the objective of our thermal-aware scheduling algorithm is to minimize the number of times that voltage scaling down in the system. To achieve this goal, our thermal-aware scheduling algorithm has to minimize the peak temperature of each core in the system. As mentioned in Section I, in order to solve the scalability issue posed by large-scale many-core systems, we proposed a decentralized thermal-aware scheduling algorithm. In this section, we present an overview of the decentralized thermal-aware scheduling flow with the reasoning of our design.

A. Design Challenges

As discussed in Section I, the key idea of the decentralized thermal-aware scheduling algorithm is to divide the processor cores in the system into clusters to avoid the communication and computation bottleneck. Then the problem of scheduling is performed in a hierarchical manner. At the global level, a suitable cluster is selected to execute the application and at the cluster level, the mapping and scheduling of the application is performed.

The main challenges of the design of the decentralized scheduling algorithm lie in two parts. The first challenge is how to best balance the workload between the global level scheduling and the cluster level scheduling such that the monitoring traffic can be minimized while maximizing the peak power reduction under the deadline constraint. This challenge can be transformed into a problem of designing proper cluster selection scheme. Whether a cluster fulfills the deadline constraint of an application and how the application affects the temperature of each core in the cluster need to be evaluated before selecting the proper cluster to execute the application. The previous decentralized power-aware scheduling algorithm, ADAM [15], evaluates the application scheduling for all the clusters at the global level although the detailed scheduling is performed inside the cluster. It may still incur high computation load in the global agent and also require relatively large amount of monitoring traffic to enable a precise evaluation. Another extreme solution is to let each cluster give a schedule of the application and let the global agent select the best one. However, in reality, many applications have task graphs containing a large number of nodes. For example, the task graph of SPEC Fpppp contains over 300 nodes [21]. Even simple heuristic-based task graph scheduling algorithms have the complexity growth rate as a quadratic function of the task graph size, as a result, the "true" distributed solution could cause a huge waste of computation time and power in a large-scale many-core system. In addition, the communication cost of sending the large task graphs to each cluster agent can also be an bottleneck for the performance and power consumption of the system.

The second challenge is how to dynamically adjust the cluster size to best accommodate the application. The cluster size has important impacts on the scheduling results in terms of performance and monitoring traffic. Increasing the cluster size improves the possibility of finding better scheduling results inside a cluster due to the increase in the search space. On the other hand, it also increases the computing complexity correspondingly as well as the monitoring traffic since the information collected from cores needs to travel a longer distance. If the cluster size is larger than necessary, the computing and communication overhead may exceed the benefits on the performance improvement. However, if the cluster size is too small, it may not be able to execute an application by its deadline. To make things worse, realistic applications sometimes vary significantly in sizes, which makes fixed-size clusters inefficient. Another issue associated with cluster size is posed by VFIs. When the cluster contains cores from multiple VFIs, the heterogeneity of the voltage and frequency

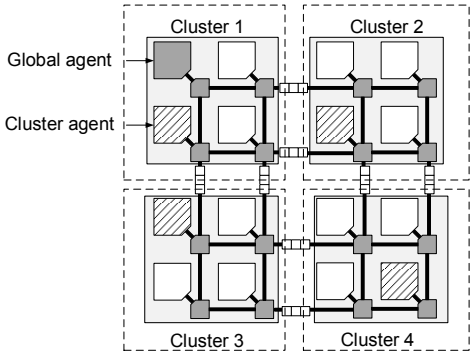


Fig. 2: Initial cluster devision and agent deploying in a VFI-based 16-core system

level and the synchronization of message transfer must be carefully considered.

B. Proposed Decentralized Approach

To address the above challenges, we have the following three key designs.

Clusters and Agents Structure: To facilitate the decentralized thermal-aware scheduling algorithm, we propose a hierarchy of software agents which run on the cores in the system. On the top level of the hierarchy, there is a global agent which manages all the agents in the system. The global agent adjusts the cluster size and selects the cluster for the application. In each cluster, there is a cluster agent responsible for scheduling the task graph inside the cluster. The cluster agents need to collect the processor core status information inside the cluster for the detailed scheduling. The cluster agents communicate with each other to exchange the temperature of processors on the cluster boarder in order to estimate the temperature of each core according to Eq. (3) in Section II.

In VFI-based many-core systems, it is better for a cluster to contain complete VFIs. This is because the when the voltage and frequency level of a VFI changes at run-time, all the cores within the VFI are affected. If the cores of a VFI belong to several different clusters, all the cluster agents that are affected by the DVFS should take actions. However, if each VFI only belongs to one cluster, the overhead of dealing with DVFS could be significantly reduced. In this study, we assume that in the initial state, each cluster only contain one individual VFI. The case where each cluster contains a few VFIs is discussed later in this section. Fig. 2 shows the cluster division and the deploying of the agents in a many-core system.

This hierarchical design brings the benefits from two aspects. Firstly, using cluster limits the problem size and efficiently reduces the complexity of the algorithm and also significantly reduces the monitoring traffic of the system. Secondly, clusters also reduce the routing paths of the inter-task communication. By carefully adjusting the cluster size and designing the cluster selection algorithm, we can efficiently balance the workload between the global agent and cluster agents to sustain the system scalability while achieving high-quality scheduling results in terms of the temperature reduction and performance.

Cluster Selection: As discussed in Section III-A, the first challenge of the decentralized scheduling algorithm is to achieve a distributed and efficient assignment and scheduling of task graphs with high quality and low monitoring traffic. As pointed out in Section III-A, selecting cluster only in global agent loses the accuracy of the resource distribution in each cluster and may also cause communication congestion. On the other hand, the overhead of scheduling the applications in each cluster could be overwhelming. In the proposed algorithm, we use a compressed data structure, called *processor load table* (PLT), to summarize the requirements on the execution time of the application for individual cores in a cluster. In cluster selection, the global agent send the PLT to each cluster and each cluster agent estimates whether the cluster could meet the deadline of the application. The cluster agents also evaluate the temperature conditions inside each cluster. Based on the evaluation of the cluster agents, the global agent picks the best cluster to host this application. By doing so, we can efficiently reduce the overhead of the "true" distributed solution without losing the information of the status of each core.

The PLT can be built by scheduling the task graph to a virtual cluster which is a logical abstraction of the computing resources contained by the real clusters. The scheduling result in the virtual cluster binds the tasks in the application to a limited number of virtual cores. From the result, we summarize the computing resource demanded by each virtual core. In this way, using PLT instead of task graphs could significantly reduce the monitoring traffic caused by cluster selection.

Based on this idea, we propose a three-step procedure for cluster selection algorithm. First, the global agent generates the PLT according to the virtual cluster size. Then the PLT is sent to the cluster agents to evaluate the thermal impact and deadline constraint of the application according to the run-time status of the cores in the cluster. Finally, the evaluation results are sent back to the global agent to pick the best cluster.

Cluster Merging: As mentioned before, in this study we assume that in initial state, each cluster contains a complete VFI. When scheduling a new application, the global agent first checks whether the cluster contains sufficient number of core to meet the deadline of the application. If the global agent finds out that the cluster size is not large enough, it initiates the cluster merging process. In the process, clusters are merged with pairs to form larger clusters. We note that this merging is not permanent, but only for one present application. In the previous method [15], the cluster merging is permanent and the large cluster only reduces when the other cluster "eats" its cores. It helps to reduce the merging overhead when the large applications come in a row, however, it not only requires more monitoring traffic on boundary control, but also generates hot-spot when all the large applications are mapped into the same large cluster. Therefore, we assume that the merging of clusters only stands valid during the scheduling of one application and there can be different ways of merging for different application to achieve the most peak temperature reduction.

We note that the initial size of the cluster is an important factor which affects the efficiency of the cluster merging. If the cluster size is too small to meet the deadline requirement of most applications, cluster merging would be frequently invoked and generate large amount of overhead. On the other hand, if the cluster size is too large for most of the applications,

the computing complexity of the scheduling algorithms would increase as well as the monitoring traffic. In Section V, we experimentally evaluates the impact of initial cluster size and the efficiency of cluster merging.

C. Overview of the Scheduling Flow

Our cluster-based decentralized thermal-aware scheduling algorithm contains five steps as shown in Fig. 3.

First, in *deadline assignment*, the global agent assigns a deadline to each individual task in the application. It is because that when there are multiple applications running simultaneously in the many-core system, our scheduling algorithm has to decide the execution order of the tasks from different applications. Hence, a deadline assignment step at the beginning of the scheduling flow is needed to assign a deadline to each individual task according to the soft deadline of the application. If the VFIs in the system are not running at the same frequency level, two set of deadlines should be assigned to the tasks, where each set of deadline is computed according to a different frequency level.

Second, in *pre-scheduling*, the global agent schedules the application to the virtual cluster and generates the PLT which represents the resources requirement of the application under the deadline. If the global agent finds out that the cluster size is not large enough to meet the timing constraint, it initiates the cluster merging process. Again, if the VFIs of the system are not running at the same frequency level, the pre-scheduling should be performed twice according to the different frequency levels. Two different PLTs of the application should also be generated accordingly.

The third step includes two sub-steps performed by the global agent and the cluster agents, respectively. First, in cluster assessment, the cluster agents check if the clusters could guarantee the deadline of the application and how the application affect the power distribution in the clusters. These assessments are based on the PLT of the application. Each cluster agent receives the PLT that is coherent with the frequency level of the cluster. For clusters that contain multiple VFIs, the frequency level of the cluster is assumed to be lowest frequency level of the VFIs. Such assumption could reduce the workload of pre-scheduling, otherwise the global agent should generate a different pre-scheduling for each different type of clusters. Then in cluster assignment, the global agent selects the best cluster for the application according to the cluster assessment results sent by the cluster agents.

In the fourth step, named by *core binding*, the cluster agent of the chosen cluster assigns the tasks of the application to the cores in the cluster. Finally, in *run-time scheduling*, each core uses the *earliest deadline first* (EDF) algorithm to decide the execution order of the tasks during run-time.

IV. Algorithm Description

This section presents the detailed algorithm of each step in our decentralized scheduling flow.

For the clarity of discussion, we use an example in this section to help introduce the thermal-aware scheduling algorithm. The example is shown in Fig. 4. In the example, an application with 8 tasks needs to be scheduled to the 16-core system which is divided into four equal-sized clusters. In the initial state,

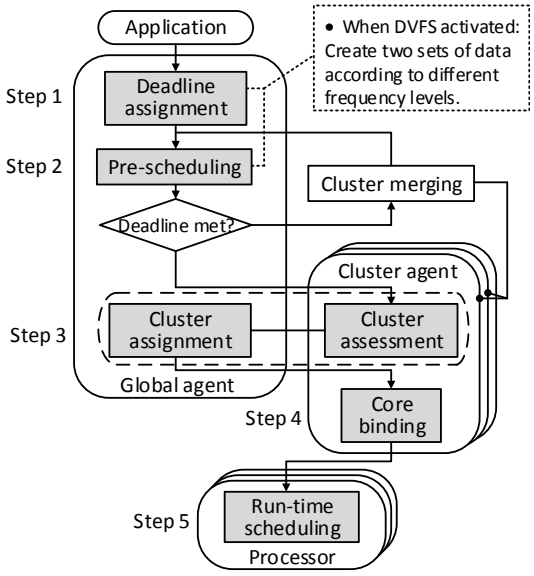
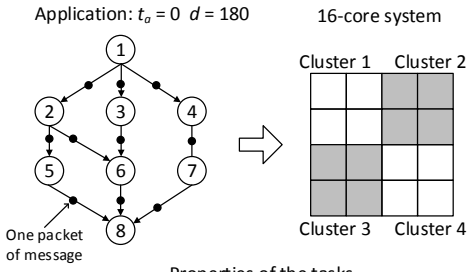


Fig. 3: The flow of our cluster-based decentralized thermal-aware scheduling algorithm



Properties of the tasks

Task	1	2	3	4	5	6	7	8
Length	20	30	30	40	40	40	35	20
Power	5	10	10	5	5	10	5	5

Fig. 4: The beginning example: an application with eight tasks is going to be scheduled to a 16-core system with four clusters.

each cluster contain one VFI of the system. The task graph of the application shows the inter-dependence relationship of the tasks. The dots on the edges of the task graph represent the size of messages transferred between tasks, where each dot stands for one packet of data. The properties of the tasks, including task length and average power consumption in unit time, are shown in the table below. In the table, properties are for the tasks running at the higher frequency level for the system. For the lower frequency level, the task length and power can be scaled according to the value of f_H and f_L. Without specification, all the examples used in the rest part of this section are assumed for the higher frequency level.

A. Step 1: Deadline Assignment

In our cluster-based thermal-aware scheduling algorithm, deadline assignment serves two purposes. First, it provides information of the timing constraints for an application when generating the processor load table. Second, it provides each task with an unique deadline which is required by the in-processor run-time scheduling. As a result, the deadline assigned to each task must meet the following condition: when the deadline of each task is met, the deadline of the entire

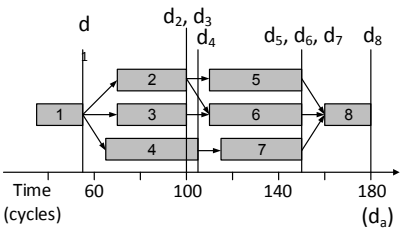


Fig. 5: Deadline assignment result of the beginning example

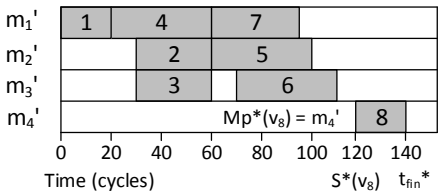


Fig. 6: The Gantt chart of example application generated in pre-scheduling

application must be met. If this condition is not satisfied, the assigned deadline could lead to the violation of the timing constraints of the applications.

There are various kinds of deadline assignment algorithms proposed in previous studies with different objectives [27], [28]. In our work, the only requirement of the deadline assignment algorithm is to make sure the timing constraint of the application is not violated. Therefore, we adopt the simple as late as possible (ALAP) algorithm, which fulfills our requirement with low complexity. In worst case, the running time of the algorithm is $O(|V|+|E|)$, where $|V|$ is the number of tasks in the task graph and $|E|$ is the number of edges. Fig. 5 shows the result of the deadline assignment of the example.

If some of the VFIs in the system are running at the lower frequency level f_L, the deadline assignment should be performed again to generate a new set of deadlines for the lower execution speed. We note that the deadlines for the lower frequency level should be no less than the deadlines for the higher frequency level.

B. Step 2: Pre-scheduling in Virtual Cluster

In pre-scheduling, the global agent schedules the applications to a virtual cluster which is of the same size of the real cluster in the many-core system. Each pre-scheduling result is used to generate the PLT, which is the summary of the resource demand of the application. If the pre-schedule cannot meet the deadline of the application, cluster merging is invoked to increase the cluster size.

Virtual Cluster: The virtual cluster is composed of a set of virtual cores which are logical abstractions of the real cores in the system. In the view of the global agent, a virtual core has the same execution speed and power consumption as a real core. On the other hand, the virtual cores do not have the run-time properties of the real cores including the temperature and the task queues which store the previously assigned tasks. We denote the virtual cluster as M', while $m'_i \in M'$ denotes each virtual core in the virtual cluster.

Pre-scheduling Algorithm: In designing the pre-scheduling algorithm, we set the objective of the algorithm as minimizing the execution time (makespan) of each application. Minimizing the makespan of an application in pre-scheduling also reduces the actual execution time of the application during run-time due to the mechanism of the processor binding, which is discussed in Section IV-D. This leads to the following advantages. When the workload of the many-core system is high, minimizing the makespan can make the deadlines of the applications more likely to be met; and when the workload of the system is low, minimizing the makespan leaves the cores more idle time which may contribute to the cooling down of the temperature in the system. This is the reason why we choose makespan minimization as the objective of the pre-scheduling algorithm. Like the deadline assignment, if the VFIs in the system are not running at a same frequency level, different pre-schedules with respect to the different frequency levels should be produced by the global agent.

Since task graph scheduling problems are NP-complete [29], we adopt an efficient heuristic-based algorithm as the pre-scheduling algorithm for the sake of run-time use. The algorithm is called highest level first with estimated time (HLFET) algorithm [29]. The scheduling results can be expressed by the following two functions. Firstly, the mapping function for the pre-schedule, denoted as $Mp^*(v_i)$, defines which virtual core each task mapped to. Secondly, the scheduling function for pre-scheduling, denoted as $S^*(v_i)$, defines the starting time for each task. We also denote the finishing time for the pre-schedule as t^*_{fin}. We note that in pre-scheduling, the communication delay of the tasks are computed by Eq. (1) proposed in Section II-A. The running time of the algorithm is $O(|M'| \times |V|)$, where $|M'|$ is the size of the virtual cluster and $|V|$ is the number of tasks contained by the task graph. Fig. 6 shows pre-scheduling result of the beginning example, which is shown in a Gnatt chart. In the example, we use task 8 to show the definition of Mp^*, S^* and t^*_{fin}. Since task 8 is mapped to virtual core m'_4, therefore there is$Mp^*(v_8)$, and the start time of v_8 is $S^*(v_8) = 120$. Finally, the finish time of the pre-schedule is at the 140^{th} cycle.

Cluster Merging: It is possible that the pre-scheduling result does not meet the deadline constraint of the application. In such case, we consider the problem is caused by insufficient number of cores in each cluster, since the pre-scheduling algorithm already aims at minimizing the makespan of the application. As mentioned in Section III, in the case, cluster merging is invoked to solve the problems. In this study, we use pre-defined cluster merging policies. The cluster merging policy specifically defines which clusters should be merged into one cluster. Using pre-defined policies could avoid generating additional monitoring traffic during cluster merging. We use an example to explain how cluster merging works. For instance, in Fig. 7, a 16-core system is initially divided into four 4-core clusters. The merging policy is also illustrated in Fig. 7. When cluster merging is invoked for the first time, the original cluster 1 and 2 are merged together into an 8-core cluster (cluster 5), and cluster 3 and 4 are merged into cluster 6. Then the global agent repeats pre-scheduling for the application with the new cluster size. If the 8-core cluster still cannot meet the requirement, then cluster 5 and 6 should be merged together into cluster 7 which contains all the cores in the system.

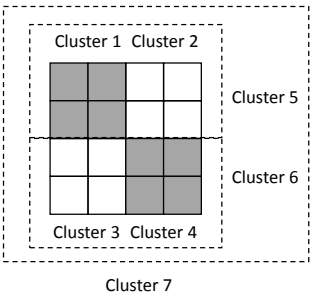


Fig. 7: Cluster merging on a 16-core system

PLT Generation: The pre-scheduling result is used to generate the PLT, which depicts the resources requirement of the cores in the virtual cluster and is used to evaluate whether a cluster fulfills the deadline constraints of the application. Therefore, to achieve this objective, the PLT should contain the following information. First, the PLT should record the total running time of each virtual processor. Second, the PLT should set a deadline to the tasks scheduled to each virtual core.

Therefore, we give the definition of PLT as follows. A processor load table, is a table of $|M'|$ entries, where $|M'|$ is the number of virtual cores. The key of each entry is the identifier of a unique virtual core, m'_i, and each entry have two elements. The first element is the execution time required by the virtual core and is denoted as l'_i. The second element is the deadline of the virtual core, denoted by d'_i. The symbols and definitions for the three elements are shown in Table I.

To ease the discussion, we use the example shown in Fig. 8 to explain l'_i and d'_i defined in Table I. To get the PLT, we first assume that the application is executed according to the pre-schedule, except for that it ends at the deadline of the application. As shown in Fig. 8, the pre-schedule is shifted to the right to end at the 180^{th} cycle. Secondly, the length of each virtual core equals to the interval between the start time of the first task mapped to the virtual core and the finish time of the last task mapped to the virtual core, including the idle time in between. As shown in Fig. 8, the length of virtual core m'_3 is 75, which equals to the sum of the length of task 3 and task 6 plus the 10 cycle interval in between. Finally, the deadline of each virtual core equals to the end execution time of the last task mapped to that virtual core. Again in Fig. 8, the deadline of virtual core m'_3 is at the 155^{th} cycle.

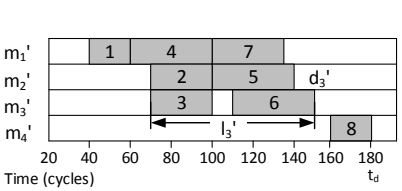
With the above definition of PLT, we can consider each virtual core as a "super task" with the length and deadline of l'_i and d'_i. When the deadline constraint of each virtual core could be met, then the deadlines of all the tasks mapped in that virtual core can be met. The influence of communication delay in PLT is also considered because in the pre-schedule, the communication between virtual cores is assumed to take the longest routing path in a real cluster. Therefore, we can use PLT to represent the requirement of execution times of the application and evaluate the available execution time in clusters according to PLT.

Using PLT instead of the original task graph significantly reduces the communication overhead of the scheduling algorithm. For example, assume that we are scheduling a SPEC

TABLE I: The three elements contained by each in the PLT

Name	Symbol	Definition
Length	l'_i	$\max_{v_j \in V'^a_i}(S^*(v_j)+l_j) - \min_{v_j \in V'_i}(S^*(v_j)+l_j)$
Deadline	d'_i	$\sum_{v_j \in V'_i} d_j$

[a] V'_i is the set of tasks that have been mapped to virtual core m'_i during pre-scheduling.



Processor load table

m_i'	l_i'	d_i'
m_1'	100	135
m_2'	70	140
m_3'	75	155
m_4'	20	180

Fig. 8: Processor load table of example application generated in pre-scheduling

Fpppp application to a many-core system with the cluster size of 16-core. The original task graph contains 334 nodes and 1145 edges [30], but the PLT only contains 16 entries. The reduction in the data for transmission is over 90%. When the cluster size increases, the total communication overhead for transmitting PLT does not increase much. Because as the number of entries of the PLT increases, the number of copies of PLT to be sent decrease as well due to the reduction of the number of clusters.

C. Step 3: Cluster Selection

As mentioned in Section III-C, cluster selection can be further divided into two sub-steps: cluster assessment in cluster agents and cluster assignments in global agent.

1) Cluster assessment: In cluster assessment, each cluster agent receives the PLT sent by the global agent and checks the cores of the cluster in two aspects. First, the cluster checks if the cores in the cluster have sufficient running time to execute the application before its deadline (deadline assessment). Second, the cluster agent checks the power distribution in the cluster after mapping the new application (thermal assessment).

Deadline Assessment: The PLT records the total execution time and the latest deadline of the tasks assigned to each virtual core. In deadline assessment, the cluster agent uses this information to check if the real cores in the cluster could meet the demand in execution time posed by the PLT. This problem can be formally formulated as follows:

Assume that we have a cluster in the many-core system, denoted by M_c, and the cluster agent receives a PLT, denoted by G_{load}. In the cluster, each real core, denoted by m, contains a list of previously assigned tasks waiting for execution, denoted by V_{wait}. As defined in Section IV-B, the deadline of virtual core m' is d' and the length of the virtual core is l'. Assume at a moment, the tasks contained by virtual core m' in the PLT is going to be scheduled to the real core m. To guarantee the deadline constraint, the real processor must provide enough free execution time, denoted by t_f, before the deadline d', such that $t_f > l'$. If the match between m' and m satisfies the condition, we call it a feasible match, otherwise we call it an unfeasible match. Then the deadline

(a) An example of bipartite graph (b) A maximum match in the bipartite graph

Fig. 9: Maximum match problem of bipartite graph

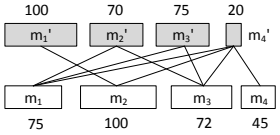


Fig. 10: An example of core matching graph

assessment problem is to find out that wether there exists an unique feasible match m in cluster M_c, for each virtual core m' in G_{load},

This problem can be transformed into a maximum matching problem of bipartite graphs, which is a well studied problem in the field of computer science. A bipartite graph is a graph whose vertices can be divided into two disjoint sets such that each edge in the graph connects two vertices from different sets. Fig. 9a shows a bipartite graph in which the gray vertices are from set A and the white vertices are from set B. In a bipartite graph, a maximum match is a subset of the edges, where each edge in the maximum match connects two unique vertices and all the vertices in the bipartite graph are connected by the edges in the maximum match. In Fig. 9a, there exists a maximum match in the bipartite graph, which is shown in Fig. 9b. The maximum match problem is to determine if there exists a maximum match in a bipartite graph.

We can transform the cluster assessment problem into a maximum matching problem of bipartite graphs as follows. First, we view the virtual cores from PLT and the real cores in the cluster as the two independent vertices sets of a bipartite graph. Second, if the free execution time provided by real core m is sufficient to run the virtual core m' before its deadline, we draw an edge between the two nodes in the graph. In this way, we form a bipartite graph which we call *core matching graph*, denoted as G_m. If there exists a maximum matching in the core matching graph, it means that the cores in the cluster are capable of executing the application before its deadline, according to previous discussion. Fig. 10 shows the core matching graph of the example. Here, the width of the nodes shows the total execution time required by the virtual cores (gray blocks) and the free execution time provided by the real cores (white blocks).

The free execution time provided by real cores is computed by Eq. 4. In the equation, V_e is a subset of V_{wait} where the deadline of each task $v \in V_e$ is earlier than d'. V_l is a subset of V_{wait} where the deadline of each task $v \in V_l$ is later than d', yet these tasks have to start execution before d' to avoid missing their deadline. As defined in Section II-B, l and d are the length and the deadline of task v respectively. If free execution time t_f is larger than the total execution time of the

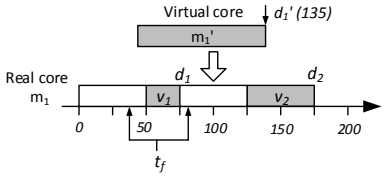


Fig. 11: Example of computing free execution time

virtual core m', then real core m is feasible for m'. Fig. 11 illustrates how to compute the free execution time of a real core.

$$t_f = d' - \sum\nolimits_{v \in V_e} l - \sum\nolimits_{v \in V_l} (l - d + d') \tag{4}$$

The maximum matching problem for bipartite graphs can be solved by a series of well-developed algorithms. In this work we select the Dinitz blocking flow algorithm [31] due to its low complexity. The Dinitz algorithm finds the maximum number of matches, denoted as N_m, for the bipartite graph G_m. If the number of matches generated by the algorithm equals to the number of nodes in PLT, it means that there are enough cores in the cluster to fulfill the deadline requirement of the application. The maximum number of matches N_m is used as the final score of deadline assessment and is sent to the global agent. Assume for a graph containing V vertices and E edges, the complexity of the Dinitz blocking flow algorithm is $O(E\sqrt{V})$.

Thermal Assessment: In thermal assessment, the cluster agent predicts the temperature of each core in the cluster and uses the highest predicted temperature as the score for thermal assessment. We use the highest predicted temperature as the thermal assessment score for two reasons. Firstly, from the highest predicted temperature we can predict whether a thermal emergency is going to happen in the cluster. Secondly, due to the thermal coupling between adjacent cores in the system, the highest predicted temperature partially reflects the temperature distribution of the cluster [32]. In other words, if the highest predicted temperature of a cluster is very high, the temperature of adjacent processors is probably also going to become very high in the future.

To predict the temperature of each core in the cluster, we use Eq. (3) proposed in Section II-C. In order to get the current temperature value of all adjacent cores, the cluster agents communicate with their adjacent neighbors to acquire the temperature of the cores locating on the cluster boarders. Since the thermal prediction step mentioned in Section II-C is quite short comparing to the total length of the application, it is not feasible to just predict the temperature of the cores for just one step. In this work, we use Eq. (3) to simulate the temperature changing in a cluster for 25 steps (0.1 s), which is set through experiments. The power trace of the thermal simulation is generated according to a EDF scheduling of the un-executed tasks inside each core, because as mentioned in Section III-C, the final step of in processor scheduling adopts the EDF algorithm. The final score of the thermal assessment, which is the highest predicted temperature of the cluster, is denoted as τ_m.

2) Cluster Assignment: After cluster assessment, cluster agents send back the scores of the deadline assignment (N_m)

Algorithm 1 Cluster selection algorithm

```
    Input: PLT, M,
    Output: C_s
 1: for all C_i ∈ M do
 2:     receive_PLT()
 3:     N_m(i) = deadline_assessment()
 4:     τ_m(i) = thermal_assessment()
 5: end for
 6: C_s = C_1
 7: for all C_i ∈ M do
 8:     if (τ_m(s) < τ_th) XOR (τ_m(i) < τ_th) then
 9:         C_s = (τ_m(s) < τ_th) ? C_s : C_i
10:     else
11:         if (N_m(s) == |M'|) XOR (N_m(i) == |M'|) then
12:             C_s = (N_m(s) == |M'|) ? C_s : C_i
13:         else if (N_m(s) == |M'|) AND (N_m(i) == |M'|) then
14:             C_s = (τ_m(s) < τ_m(i)) ? C_s : C_i
15:         else
16:             C_s = (N_m(s) > N_m(i))? C_s : C_i
17:         end if
18:     end if
19: end for
```

and the thermal assessment (τ_m) to the global agent. To find the best cluster to host the application, we propose the following principles for the global agent to make decisions. The following principles are listed according to the descending order of their priorities.

- The global agent always tries to select the cluster whose highest predicted temperature is lower than the threshold temperature to avoid thermal emergencies. ($\tau_m < \tau_{th}$)
- The global agent always tries to select the cluster which meets the deadlines of all the virtual cores in the PLT such that the deadline constraint of the application is fulfilled. ($N_m = |M'|$)
- The global agent always tries to select the cluster which meets the deadlines of more virtual cores in the PLT such that the possibility of meeting the deadline of the application is higher.
- The global agent always tries to select the cluster with lower highest predicted temperature such that the peak temperature of the system could be reduced.

The pseudo code of cluster selection is shown in Algorithm 1. The output of the algorithm is the selected cluster to host the new application, denoted as C_b. Line 1 to line 5 is the cluster assessment within each cluster and the score of the cluster assessment is N_m and τ_m as discussed before. Line 7 to line 19 is the selection process performed by the global agent. Line 8 to line 9 selects the cluster whose τ_m does not exceeds the threshold temperature. Line 10 to line 18 are for the cases when the thermal constraints of the two clusters are both met or both violated. In such cases, the global agent first picks the cluster that fulfills the deadline constraints (line 11, 12). For clusters both meet the constraints, the global agent picks the cluster with lower highest predicted temperature (line 13, 14). For clusters both miss the deadlines, the global agent picks the cluster with more feasible matches in the processor match graph (line 15,16).

D. Step 4: Core Binding

After the global agent selects the cluster to execute the application, it sends the application together with the pre-scheduling information to the cluster agent. Then the cluster agent should bind a real core to each virtual core in the pre-schedule. In core binding, the first objective is to find a mapping that meets the deadline constraints of the virtual core, and the second objective is to minimize the peak temperature of the cores. Since the algorithm is for run-time use, we design a heuristic-based algorithm which solves the problem with low complexity. The basic methodology for processor binding is to map the virtual core with highest power consumption, p', to a real core which results in a scheduling that leads to the lowest predicted temperature τ_{pred} [25]. In processor binding, we also consider the influence of communication power, which also contributes to the temperature rise in the system.

Algorithm 2 Core binding algorithm

```
    Input: PLT, M, Processor matching graph
    Output: Mp : M' → M
 1: while M ≠ ∅ do
 2:     m' = find_max_power(M')
 3:     M_f = find_feasible_processors(M)
 4:     if M_f = ∅ then
 5:         m = find_max_free_time( M)
 6:     else
 7:         compute_peak_temperature()
 8:         m = find_min_cost(M_f)
 9:     end if
10:     Mp(m') = m
11: end while
```

The pseudo code of the algorithm in shown in Algorithm 2. In the algorithm, line 2 picks the virtual cores with highest power consumption from the PLT. To find a proper real core to host the virtual core, line 3 first finds the feasible cores which meets the deadline constraint of m'. This is enabled by referring to the core matching graph generated in deadline assessment. If there does not exist any feasible core, then the virtual processor m' is bound to the real core with most free execution time where the possibility for meeting the deadline is higher. Otherwise, we use a cost function to find the best core among the feasible cores (line 8). The cost function is defined by Eq. (5), where τ_{pred} is the predicted peak temperature for the real core and E_{comm} is the communication energy and a is a tuning parameter. τ_{pred} is computed in line 7 using Eq. (3) in Section II-C. Firstly, we generate the power trace of the core according to an EDF schedule of the tasks in the real core and the virtual core. Secondly, we use Eq. (3) to estimate the temperature trace of the real core to the end of the last task in the schedule. Finally, we select the peak temperature t_{pred} from the temperature trace. E_{comm} can be computed by Eq. (6), where E_{bit} is defined in Eq. (2) in Section II-A and vol_{mapped} is the total communication volume between the virtual core which is being mapped and the virtual cores that are previously bound to real cores. After the processor binding is finished, the tasks inside each virtual core are assigned to the real core according to the mapping results.

$$cost = \tau_m + a \cdot E_{comm} \tag{5}$$

$$E_{comm} = E_{bit} \cdot vol_{mapped} \tag{6}$$

E. Step 5: Run-time In-processor Scheduling

After core binding, the cores begins to execute the tasks. In each core, an earliest deadline first algorithm is adopted to decide the execution order of the tasks. The detailed structure of the in-processor scheduler is shown by Fig. 12. Each core

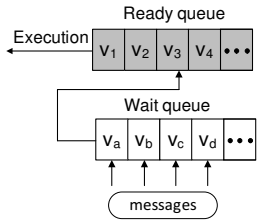


Fig. 12: The flow of run-time task scheduling

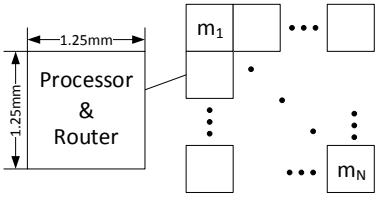


Fig. 13: The general floorplan of the many-core systems in the experiments

keeps two queues of tasks to be executed. Tasks that are in the ready queue can start execution immediately. Tasks that are in the wait queue cannot start execution until all the messages for their precedence tasks are received. Whenever a core becomes free, the EDF scheduling algorithm picks the task with an earliest deadline from the ready queue to execute on the processor. The deadline of each task has been defined during the deadline assignment step in the global agent. This scheme makes it possible for one cluster to simultaneously execute tasks of multiple applications to improve the utility rate of the cores.

V. Evaluation

In this section, we present the experimental results on the evaluation of the decentralized scheduling scheme.

A. Experiment Setup

Simulation Settings: We adopt the HotSpot thermal simulator as the experimental platform. In the experiments, we model three many-core systems with different size (64 cores, 256 cores and 1024 cores) for the sake of scalability evaluation. The cores are fixed to be $28\mathrm{nm}$ ARMv7 architecture. Fig. 13 shows the general floorplan of the many-core systems used in the experiments. The many-core systems are tile-based, where each tile contains a processer core and a router. In the floorplan, each tile is a square with the side length of $1.25\mathrm{mm}$ and all the tiles are aligned into a square matrix. Table II summarizes the thermal-related parameters for the many-core systems which is used in HotSpot. We assume that in each VFI in the many-core systems contain 16 cores and that the frequency levels for VFI are 1.9 GHz and 800 GHz.

TABLE II: Thermal-related parameters configuration

Parameter	Value
Die thickness	0.15 mm
TIM[a] thickness	0.02 mm
Silicon heat capacity	1.76×10^6 J/(m$^3 \cdot$ K)
Silicon thermal conductivity	148 W/(m $\cdot$ K)
TIM thermal conductivity	4 W/(m $\cdot$ K)

[a]thermal interface material

Benchmarks: In the experiments, we adopt the synthesised task graphs provided by the Standard Task Graph (STG) suite as the benchmarks. STG is a fair and complete benchmark suite for randomly generated task graphs and used in previous studies like [30]. However, the task graphs provided by the STG do not have power consumption, so we have to manually generate a power consumption figure for each of the tasks in the benchmarks. The power consumption figures we generated are based on the average power consumption of the programs in SPEC CPU2006 running on a $28\mathrm{nm}$ quad-core ARM Cortex 15 processor with the ARMv7 architecture. The power figures are acquired by simulation on the cycle-accurate simulator GEM5 [33]. The observed power consumption per core of SPEC CPU2006 benchmarks ranges from 0.74 W to 0.56 W. We assign power consumption to the tasks according to a uniform distribution of the values in the above range. In addition, the idle power of each core is assumed to be 0.11 W.

We combine the task graphs from four different packages of STG to form an application pool. The task graphs from the four packages respectively contain 50, 100, 200 and 300 nodes each. The task graphs with 50 nodes represents embedded applications like FFT (48 nodes). Task graphs with 100 nodes represents applications like sparse matrix solver (96 nodes). Task graphs with 200 nodes represents applications like H263E video decoder (201 nodes). Task graphs with 300 nodes represents applications like SPEC Fpppp (334 nodes). [21] In the application pool, each application contains 162.5 nodes on average. For the 64-core system, we send 16 applications to the system every 100 ms. The number of applications issued to the many-core system at one time is proportional to the number of processors contained by the system. This is to eliminate the difference in the system load caused by the different amount of computing resources in different systems.

The deadline of each application is set by Eq. (7). In the equation, d is the deadline of the application, t_a is the arriving time of the application, $l_{critical}$ is the length of the critical path of the application, namely, the minimum makespan of the application, and α is a tuning factor. We note that α should not be lower than 1, otherwise the deadline is impossible to be met. By giving α different values, we could control the tightness of the deadlines for the applications. In the experiments, we set $\alpha = 2$, such that the deadlines are neither too tight nor too loose.If the deadlines are too tight, the priority of meeting deadline constraints for our decentralized thermal-aware scheduling scheme becomes overwhelming and the search space left for the thermal management is too small. If the deadlines are too loose, then deadline constraints have little impact on the scheduling results, which could make the comparison between our algorithm and the deadline-aware algorithm meaningless.

$$d = t_a + \alpha \times l_{critical},\ \alpha \geq 1 \tag{7}$$

Comparison algorithms: Since there is no previous study directly comparable with our work, we select a number of thermal-aware scheduling algorithms in different perspectives to thoroughly evaluate our decentralized thermal-aware

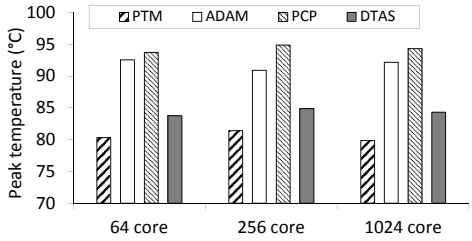


Fig. 14: Peak temperature of the many-core systems using different scheduling algorithms

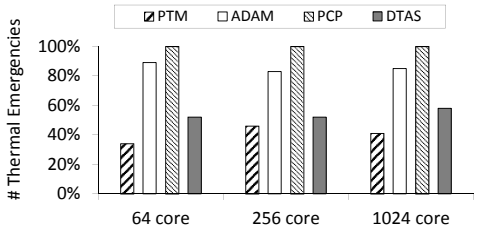


Fig. 15: Normalized number of thermal emergencies in the many-core systems using different scheduling algorithms

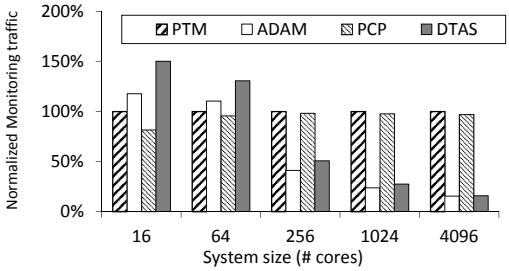


Fig. 16: Normalized monitoring traffic generated by the scheduling algoirhtms.

scheduling algorithm. First, to show the effectiveness of our algorithm in peak temperature reduction, we compare our work with a state-of-the-art centralized thermal-aware scheduling algorithm for task graph. The algorithm is called the predictive task migration algorithm [34] (denoted as *PTM*). The PTM algorithm aims at balancing the temperature distribution in many-core systems by dynamically migrating tasks from hot cores to cool cores. We use PTM algorithm for comparison is because it is the most similar study as our work. It is designed for thermal-aware task graph scheduling problem and it is also a dynamic solution. Second, to evaluate the scalability and efficiency of our work, besides the centralized thermal-aware scheduling algorithm, we also compare our work to the cluster-based decentralized scheduling algorithm, similar as ADAM [15]. ADAM is a power-aware scheduling algorithm, which adopts the cluster-based decentralization method and evaluates the cluster selection at the global agent. Thirdly, to evaluate the performance of our algorithm on meeting the deadlines, we select a deadline-aware task graph scheduling algorithm called the partial critical path algorithm [35] (denoted as *PCP*). We extend the similar approach for thermal-aware scheduling for better comparison. In the experiments, our decentralized thermal-aware scheduling algorithm is denoted as *DTAS*.

B. Performance in Thermal Management

Fig. 14 shows the comparison of peak temperature of the many-core systems without DVFS achieved by different scheduling algorithms. In Fig. 14, PCP results in highest peak temperature because it is not designed for thermal management. ADAM shows the second highest peak temperature for the same reason but when compared to PCP, the power saving feature of ADAM also slightly reduces the peak temperature. PTM achieves the lowest peak temperature in all the cases due to the global optimization. However, PTM causes 3.74% of the applications to miss their deadlines on average since the algorithm is not designed for deadline constraints. Our decentralized thermal-aware scheduling algorithm, on the other hand, guarantees the deadlines of all the applications in the experiments. The decentralized algorithm achieves peak temperature only 3.68% higher than PTM on average. In addition, when considering the communication overhead generated for each application, our decentralized algorithm significantly outperforms the centralized algorithm. In experiments, an average 34.7% reduction of communication delay and 48.9% reduction of communication power is achieved by our DTAS algorithm when compared to the centralized thermal-aware scheduling algorithm PTM.

Fig. 15 shows the normalized number of thermal emergencies appeared in the system when different scheduling algorithms are applied. In this experiments, the threshold temperature for each core is set at $75^{\circ}\mathrm{C}$. The threshold temperature is selected to avoid dark silicon problem [7]. In the figure, we can see that in all cases, PTM achieves the least number of thermal emergencies due to is global optimization, when compared to PCP, 59.7% of the thermal emergencies are reduced on average. Our DTAS algorithm achieves an average reduction of 46.3% compared to PCP. When considering the deadline constraint, PCP and ADAM cause 14.8% and 12.9% deadline miss due to the frequently happened thermal emergencies. PTM causes 7.8% deadline miss due to the lack of deadline meeting scheduling policies. Our DTAS algorithm, however, only causes 1.2% deadline miss with specific concerns for the deadline constraints under DVFS.

C. Monitoring Traffic

Fig. 16 compares the normalized total monitoring traffic generated by the scheduling algorithms. In small scale systems like the 16-core and 64-core systems, the monitoring traffic of centralized algorithms (PTM and PCP) is lower than the decentralized algorithms (ADAM and DTAS). This is due to the additional monitoring traffic generated by the communications between agents. However, when the system size reaches 256 core, significant monitoring traffic reduction can be achieved by decentralized scheduling algorithms. When compared with ADAM, our DTAS algorithms consumes slightly higher monitoring traffic due to the transfer of PLTs. When compared to centralized scheduling algorithms, our DTAS scheduling algorithm achieves up to 84.3% of monitoring traffic reduction.

D. Exploration of Initial Cluster Size

In this section, we explore the influence of the initial-cluster size on the decentralized thermal-aware scheduling algorithms.

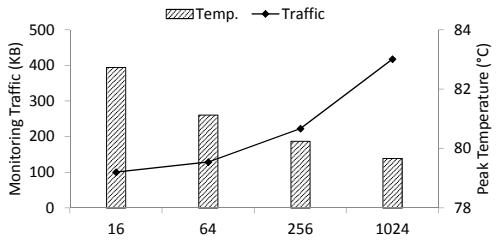


Fig. 17: Peak temperature and monitoring traffic under different initial cluster size

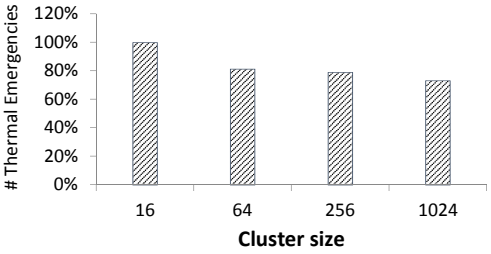


Fig. 18: Normalized number of thermal emergencies under different initial cluster size

In the experiments, the task graphs in the benchmarks are scheduled to the 1024-core system by our decentralized thermal-aware scheduling algorithm. We use four setting for the initial cluster size: 16-core, 64-core, 256-core and 1024-core, respectively. In the initial state, each cluster is set to be a square array of cores in the system. This is to prevent different shapes of the clusters from affecting the results of the experiments.

Fig. 17 shows the peak temperature and monitoring traffic of the system without DVFS. As the initial cluster size increases, the peak temperature of the system shows a slight reduction. As we have discussed previously, this is due to the increase in the search space of the algorithm. However, we can also observe a significant increase in the monitoring traffic with cluster size increase.

Fig. 18 shows the normalized number of thermal emergencies happened on the system with DVFS presented. Similar as the trend shown in Fig. 17, the number of thermal emergencies decreases as the cluster size grows. This is due to the search space for the processor binding algorithm increases and the possibility of finding a more thermal efficient schedule also increases.

E. Evaluation of Cluster Merging

To further demonstrate the benefits of the cluster merging, we conduct an experiment of our decentralized thermal-aware scheduling algorithm without the cluster merging technique (denoted as DTAS (no merge)). Whether or not the pre-scheduling results meet the deadline constraints, the global agent sends the PLT to the cluster agents to continue the scheduling process. The experiment is carried out on the 1024-core system with the initial cluster size of 16-core.

Table III shows the experimental results. First, the scheduling algorithm with cluster merging significantly outperforms the one without cluster merging in deadline miss rate. This is

TABLE III: Performance evaluation of cluster merging

Scheduling algorithm	DTAS	DTAS (no merge)
Deadline miss rate	0	12.97%
Peak temperature ($^\circ C$)	82.42	84.91
Avg. montr. traffic (KB)	573.45	496.47

the motivation of implementing the cluster merging technique: to provide more computing resources for large applications to meet their deadline constraint. Second, with the cluster merging technique, the decentralized thermal-aware scheduling algorithm achieves lower peak temperature in the system. This is because cluster merging results in larger clusters in the system, which could lead to potential improvement in the scheduling results. Finally, as the overhead of cluster merging, the monitoring traffic of the system increases by 15.5%. This overhead is acceptable considering the improvement in deadline miss rate, more importantly with the gain of lower temperature.

VI. Related Work

With the development in technology scaling, microprocessors have become more vulnerable to thermal failures due to the ever increasing power density. In order to improve the reliability of the microprocessors, as well as to reduce the cost spent on cooling systems, dynamic thermal management (DTM) has become an important focus of the research in microprocessors. Previous studies have proposed various kinds of DTM techniques, which can be classified into two main categories [36]. The *core-throttling* techniques directly control the power consumption of the processors by affecting the execution speed of the hardware, which usually adopt DVFS scheme [23]. The *non-core-throttling* techniques, which are mainly composed of thermal-aware scheduling algorithms, are pure software-level DTM solutions. In general, core-throttling techniques outperform non-core-throttling techniques in control accuracy and response speed. However, non-core-throttling techniques have much lower implementation cost and do not affect the performance of the processors.

When dealing with different kinds of workloads, the thermal-aware scheduling algorithms adopt different methodologies. Many of the studies in thermal-aware scheduling algorithms are proposed for independent tasks. Recent studies usually solve the thermal management problems considering different kinds of architecture-level features. For instance, Donald *et al.* [37] proposed a scheduling algorithm for simultaneous multi-threading (SMT) processors, which balances the power consumption of the processors by simultaneously running high-power and low-power tasks. Khdr *et al.* [38] proposed a thermal constrained resource management solution for tasks with both instruction level parallelism and thread level parallelism.

As parallel computing becomes pervasive, there is growing interest on the thermal-aware scheduling algorithms using task graphs. Thermal-aware scheduling algorithms for task graphs can be classified by the type of scheduling algorithms adopted [39]. Algorithmic schemes are capable of producing optimal or near-optimal scheduling solutions through high-complexity algorithms. Puschini *et al.* [12] adopted gaming theory to reduce the peak temperature of NoC systems. Conskun *et al.* [34] formulated the problem as a mixed integer linear

programming problem (ILP). The study not only provided the formal ILP solution which guarantees optimal results, but also provided a simplified heuristic-based solution for dynamic use. Heuristic scheduling algorithms are fast enough for on-line scheduling and carefully designed heuristics can achieve comparable performance as the algorithmic algorithms under most real-life workloads. Hung *et al.* [8] proposed a power-balancing heuristic which effectively reduces the peak temperature in multi-core systems. Coskun *et al.* [34] proposed a heuristic to mitigate the temperature gradient between adjacent cores in the system. The algorithm proposed in that study required on-line thermal simulation to estimate temperature difference between cores. Khdr *et al.* [40] proposed a scheduling algorithm combined with DVFS to achieve multiple objectives in thermal management. Their solution not only reduces the peak temperature of the system, but also minimizes the temperature differences in adjacent cores.

However, all the above mentioned algorithms are centralized techniques, which are performed by a centralized scheduler in the system,. As the system scale grows larger, these algorithms are highly susceptible to failure due to the increasing communication. To address the scalability issue, several decentralized thermal-aware scheduling techniques have been studied [15]–[17], [41]. Ebi *et al.* [41] transformed the thermal-aware scheduling problem into an economic trading model. In the economic model, each core acted like a trading agent to sell out or buy in power budgets. Then the scheduler assigned tasks to each core according to the power budgets. In [16], they further proposed a hierarchical solution for assigning power budgets to the cores in NoC for thermal management. However, the studies only considered independent tasks. In a closely related work, Faruque et al. [15] proposed a decentralized task graph mapping algorithms for heterogeneous many-core systems. Ismail *et al.* [17] proposed a model checker to formally verify the performance of decentralized thermal aware scheduling algorithms. However, this work was not designed for thermal management, instead it aimed at minimizing communication overhead.

VII. Conclusions

In this paper, we propose a decentralized thermal-aware scheduling algorithm for large-scale many-core systems. By dividing the system into clusters, our algorithm gains high scalability and efficiently reduces the peak temperature of the many-core systems as well as meets all the deadlines compared to scheduling algorithms which aim at only performance. When compared to centralized scheduling algorithms, our work shows significant reduction up to 84.3% in the monitoring traffic overhead and computing complexity with comparable peak temperature reduction in the large-scale multi-core systems.

Acknowledgments

This work is partly supported by MoE AcRF Tier 2 grants (MOE2012-T2-2-067 and MOE2012-T2-1-126) in Singapore, and by A*Star - SERC Public Sector Funding (PSF) of Singapore under Grant No. 1121202015.

References

[1] H. Iwai, "Roadmap for 22nm and beyond," Tech. Rep. 7, 2009.
[2] S. Borkar, "Thousand core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conf.* ACM, 2007, pp. 746–749.
[3] G. M. Link and N. Vijaykrishnan, "Thermal trends in emerging technologies," in *Proceedings of the 7th Int. Symp. on Quality Electronic Design*. IEEE Computer Society, 2006, pp. 625–632.
[4] G. Gielen, P. De Wit, E. Maricau, J. Loeckx, J. Martín-Martínez, B. Kaczer, G. Groeseneken, R. Rodríguez, and M. Nafría, "Emerging yield and reliability challenges in nanometer cmos technologies," in *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2008, pp. 1322–1327.
[5] H. Esmaeilzadeh and *et al.*, "Dark silicon and the end of multicore scaling," in *Computer Architecture (ISCA), 2011 38th Annual Int. Symp. on*. IEEE, 2011, pp. 365–376.
[6] M. Shafique, S. Garg, J. Henkel, and D. Marculescu, "The eda challenges in the dark silicon era," in *Design Automation Conf. (DAC), 2014 51st ACM/EDAC/IEEE*. IEEE, 2014, pp. 1–6.
[7] J. Henkel, H. Khdr, S. Pagani, and M. Shafique, "New trends in dark silicon," in *Design Automation Conf. (DAC), 2015 52nd ACM/EDAC/IEEE*. IEEE, 2015, pp. 1–6.
[8] W.-L. Hung, Y. Xie, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "Thermal-aware task allocation and scheduling for embedded systems," in *Proceedings of the conference on Design, Automation and Test in Europe-Volume 2*. IEEE Computer Society, 2005, pp. 898–899.
[9] J. Choi and *et al.*, "Thermal-aware task scheduling at the system software level," in *Proc. of the 2007 international symposium on Low power electronics and design*. ACM, 2007, pp. 213–218.
[10] C.-L. Chou and R. Marculescu, "Incremental run-time application mapping for homogeneous nocs with multiple voltage levels," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2007 5th IEEE/ACM/IFIP Int. Conf. on*. IEEE, 2007, pp. 161–166.
[11] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Temperature-aware mpsoc scheduling for reducing hot spots and gradients," in *Proc. of the 2008 Asia and South Pacific Design Automation Conf.* IEEE Computer Society Press, 2008, pp. 49–54.
[12] D. Puschini, F. Clermidy, P. Benoit, G. Sassatelli, and L. Torres, "Temperature-aware distributed run-time optimization on mp-soc using game theory," in *Symp. on VLSI, 2008. ISVLSI'08. IEEE Computer Society Annual*. IEEE, 2008, pp. 375–380.
[13] F. Zanini, D. Atienza, L. Benini, and G. De Micheli, "Multicore thermal management with model predictive control," in *Circuit Theory and Design, 2009. ECCTD 2009. European Conf. on*. IEEE, 2009, pp. 711–714.
[14] U. Y. Ogras, R. Marculescu, P. Choudhary, and D. Marculescu, "Voltage-frequency island partitioning for gals-based networks-on-chip," in *Proc. of the 44th annual Design Automation Conf.* ACM, 2007, pp. 110–115.
[15] A. Faruque, M. Abdullah, R. Krist, and J. Henkel, "Adam: run-time agent-based distributed application mapping for on-chip communication," in *Proc. of the 45th annual Design Automation Conf.* ACM, 2008, pp. 760–765.
[16] T. Ebi, D. Kramer, W. Karl, and J. Henkel, "Economic learning for thermal-aware power budgeting in many-core architectures," in *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2011 Proc. of the 9th Int. Conf. on*. IEEE, 2011, pp. 189–196.
[17] M. Ismail, O. Hasan, T. Ebi, M. Shafique, and J. Henkel, "Formal verification of distributed dynamic thermal management," in *Proc. of the Int. Conf. on Computer-Aided Design*. IEEE Press, 2013, pp. 248–255.
[18] L. Benini and G. De Micheli, "Networks on chips: a new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.
[19] S. R. Vangal and *et al.*, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *Solid-State Circuits, IEEE J. of*, vol. 43, no. 1, pp. 29–41, 2008.
[20] J. Held, J. Bautista, and S. Koehl, "From a few cores to many: A tera-scale computing research overview," *white paper, Intel*, 2006.
[21] W. Liu and et al., "A noc traffic suite based on real applications," in *ISVLSI*, 2011, pp. 66–71.
[22] B. Goel and *et al.*, "Portable, scalable, per-core power estimation for intelligent resource management," in *Green Computing Conf., 2010 Int.* IEEE, 2010, pp. 135–146.
[23] Y. Wang, K. Ma, and X. Wang, "Temperature-constrained power control for chip multiprocessors with online model estimation," in *ACM SIGARCH computer architecture news*, vol. 37, no. 3. ACM, 2009, pp. 314–324.

[24] D. Brooks and M. Martonosi, "Dynamic thermal management for high-performance microprocessors," in *High-Performance Computer Architecture, 2001. HPCA. The Seventh Int. Symp. on*. IEEE, 2001, pp. 171–182.

[25] K. Stavrou and P. Trancoso, "Thermal-aware scheduling: A solution for future chip multiprocessors thermal problems," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conf. on*. IEEE, 2006, pp. 123–126.

[26] X. Zhou, J. Yang, Y. Xu, Y. Zhang, and J. Zhao, "Thermal-aware task scheduling for 3d multicore processors," *Parallel and Distributed Systems, IEEE Tran. on*, vol. 21, no. 1, pp. 60–71, 2010.

[27] J. Yu, R. Buyya, and C. K. Tham, "Cost-based scheduling of scientific workflow applications on utility grids," in *e-Science and Grid Computing, 2005. First Int. Conf. on*. IEEE, 2005, pp. 8–pp.

[28] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *Proc. of 2011 Int. Conf. for High Performance Computing, Networking, Storage and Analysis*. ACM, 2011, p. 49.

[29] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys (CSUR)*, vol. 31, no. 4, pp. 406–471, 1999.

[30] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *J. of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.

[31] Y. Dinitz, "Dinitzalgorithm: The original version and evens version," in *Theoretical Computer Science*. Springer, 2006, pp. 218–240.

[32] K. Skadron, M. R. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan, "Temperature-aware microarchitecture: Modeling and implementation," *ACM Tran. on Architecture and Code Optimization (TACO)*, vol. 1, no. 1, pp. 94–125, 2004.

[33] N. Binkert and *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[34] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross, "Static and dynamic temperature-aware scheduling for multiprocessor socs," *Very Large Scale Integration (VLSI) Systems, IEEE Tran. on*, vol. 16, no. 9, pp. 1127–1140, 2008.

[35] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[36] J. Donald and M. Martonosi, "Techniques for multicore thermal management: Classification and new exploration," in *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2. IEEE Computer Society, 2006, pp. 78–88.

[37] ——, "Leveraging simultaneous multithreading for adaptive thermal control," in *Second Workshop on Temperature-Aware Computer Systems*. Citeseer, 2005.

[38] H. Khdr, S. Pagani, M. Shafique, and J. Henkel, "Thermal constrained resource management for mixed ilp-tlp workloads in dark silicon chips," in *Proc. of the 52nd Annual Design Automation Conf.* ACM, 2015, p. 179.

[39] H. F. Sheikh, I. Ahmad, Z. Wang, and S. Ranka, "An overview and classification of thermal-aware scheduling techniques for multi-core processing systems," *Sustainable Computing: Informatics and Systems*, vol. 2, no. 3, pp. 151–169, 2012.

[40] H. Khdr, T. Ebi, M. Shafique, H. Amrouch, and J. Henkel, "mdtm: multi-objective dynamic thermal management for on-chip systems," in *Proc. of the conference on Design, Automation & Test in Europe*. European Design and Automation Association, 2014, p. 330.

[41] T. Ebi, M. Faruque, and J. Henkel, "TAPE: Thermal-aware agent-based power econom multi/many-core architectures," in *Computer-Aided Design-Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM Int. Conf. on*. IEEE, 2009, pp. 302–309.

Yingnan Cui Yingnan Cui received his Bachelor degree in Automation from Harbin Institute of Technology (2006-2010). He is now a Ph.D. student in School of Computer Engineering, Nanyang Technological University, Singapore (2010-). His research topic focuses on run-time thermal management of multi-core systems.

Wei Zhang Dr. Wei Zhang received her Ph.D. degree in Electrical Engineering from Princeton University. She joins Hong Kong University of Science and Technology in 2013 and establishes Reconfigurable System Lab. She was an assistant professor in School of Computer Engineering at Nanyang Technological University, Singapore (2010-2013). She is a co-investigator of Singapore-MIT Alliance for Research and Technology and works on low-power electronics.

Vivek Chaturvedi Dr. Vivek Chaturvedi is currently working as a research fellow at Nanyang Technological University, Singapore. Dr. Chaturvedi graduated with a Ph.D. in Electrical Engineering from Department of Electrical and Computer Engineering, Florida International University, Miami in 2013. He received his M.S. degree from Syracuse University, NY in 2008.

Bingsheng He Dr. Bingsheng He received the bachelor degree in computer sicence from Shanghai Jiao Tong University (1999-2003), and the Ph.D. degree in computer science in Hong Kong University of Science and Technology (2003-2008). Dr. He is an assistant professor in School of Computer Engineering of Nanyang Technological University, Singapore. His research interests are high performance computing, cloud computing, and database systems.