Distributed thermal-aware task scheduling for 3D network-on-chip

Cui, Yingnan; Zhang, Wei; Yu, Hao

2012

Cui, Y., Zhang, W., & Yu, H. (2012). Distributed thermal-aware task scheduling for 3D network-on-chip. 2012 IEEE 30th International Conference on Computer Design (ICCD 2012), 494-495.

https://hdl.handle.net/10356/99911

https://doi.org/10.1109/ICCD.2012.6378690

Distributed Thermal-Aware Task Scheduling for 3D Network-on-Chip

Yingnan Cui, Wei Zhang
School of Computer Engineering
Nanyang Technological University
Singapore
Email: ycui1@e.ntu.edu.sg, zhangwei@ntu.edu.sg

Hao Yu
School of Electrical and Electronic Engineering
Nanyang Technological University
Singapore
Email: haoyu@ntu.edu.sg

***Abstract*—The development of 3D integration technology significantly improves the bandwidth of network-on-chip (NoC) system. However, the 3D technology-enabled high integration density also brings severe concerns of temperature increase, which may impair system reliability and degrade the performance. Task scheduling has been regarded as one effective approach in eliminating thermal hotspot without introducing hardware overhead. However, centralized thermal-aware task scheduling algorithms for 3D-NoC have been limited for incurring high computational complexity as the system scale increase. In this paper, we propose a distributed agent-based thermal-aware task scheduling algorithm for 3D-NoC which shows high scheduling efficiency and high scalability. Experimental results have shown that when compared to the centralized algorithms, our algorithm can achieve up to 13 $^{\circ}C$ reduction in peak temperature of the system without sacrificing performance.**

I. Introduction

3D-NoC architecture is a feasible design to be used in future high performance computing systems because of its smaller area, lower signal delay and higher bandwidth [1]. However, the growth in power density and heat dissipation makes 3D-NoC system more easily get overheated and harms the reliability of the system [2]. Dynamic thermal-aware scheduling is an efficient solution for thermal management on 3D-NoC without incurring much implementation cost. However, previous scheduling algorithms for NoC systems like [3], [4] are all designed in the centralized manner, which limits the scalability of these algorithms. As the system scale increases, the complexity and overhead of centralized scheduling algorithms will rise rapidly and the efficiency in thermal management may also drop significantly. To overcome this problem, we proposed the first distributed thermal-aware task scheduling algorithm for 3D-NoC systems. The algorithm organizes the processors of the 3D-NoC into clusters and the scheduling of an application will be performed within each cluster simultaneously. All the schedule procedures are handled by three kinds of software agents. In experiments, our distributed algorithm not only showed higher scalability but also achieved lower peak temperature when compared to centralized scheduling algorithms.

II. Agent-based Distributed Scheduling Scheme

In this work, we use task graphs to model the input applications. A task graph is an acyclic graph (DAG), as shown in Fig. 2(a), where the nodes represent the tasks of the application and the edges denote the precedence constraints between the tasks. Execution time and power consumption of each task as well as the amount of messages to be transmitted on each edge are assumed to be previously known.

In the scheduling algorithm, we view the processors of the 3D-NoC on three levels: processor, cluster and the system. The whole system is divided into several clusters and each cluster contains same number of processors. Three kinds of software agents are used to manage the

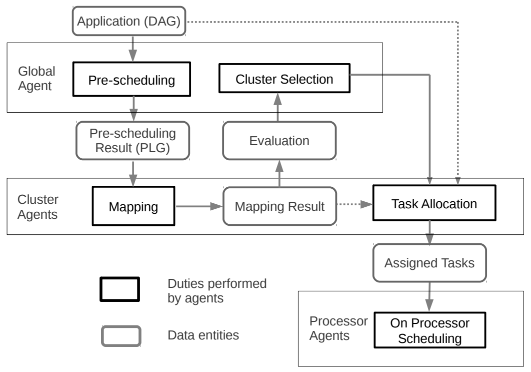


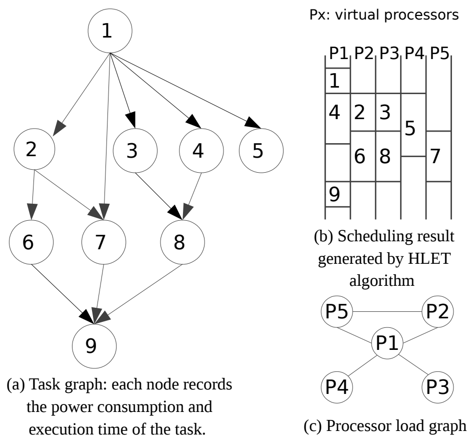
Fig. 1. The flow of the agent-based distributed scheduling algorithm.

computing resources on the three levels and perform different duties of the scheduling algorithm:

- The global agent receives an application and gives a pre-schedule to the application. It is also responsible for selecting the holding cluster for the application later.
- Cluster agents map the application onto the processors based on the pre-scheduling result given by the global agent. Each cluster agents will give an evaluation of the mapping result as the reference for the global agent to select the holding cluster for the application. After the holding processor is chosen, the agent of that cluster will load the tasks onto the processors.
- Processor agents decide the execution order of the local task. Unlike the global agent and cluster agents that run before the application starts, processor agents make decisions in between the execution of individual tasks of the application.

The scheduling flow is shown in Fig. 1. The distributed manner of this scheduling algorithm lies in the fact that the mapping of an application will be searched simultaneously in all the clusters, which greatly reduces the computing complexity of the algorithm.

The pre-scheduling in global agent is formed as the classic resource constrained DAG scheduling problem: given a set of *virtual processors* that has the same number of processors within each cluster, find a schedule of the tasks such that the execution time of the application is minimized. We adopt the the *Highest Level First with Estimated Times* (HLFET) algorithm to solve the problem [5]. This list-based algorithm schedules the tasks based on the so called *level* of each task which defines the criticality of the task. Tasks with high level need to start early, because if delayed, execution time of the whole application will be affected. Contrarily, low level tasks can be delayed for some time without affecting the execution time of the



(a) Task graph: each node records the power consumption and execution time of the task.

(b) Scheduling result generated by HLET algorithm

(c) Processor load graph

Fig. 2. Example of task graph, pre-scheduling result and PLG

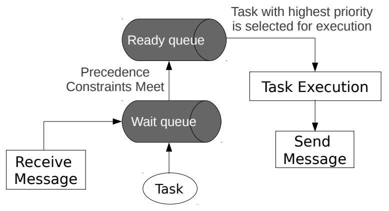

Ready queue
Task with highest priority is selected for execution
Precedence Constraints Meet
Task Execution
Wait queue
Send Message
Receive Message
Task

Fig. 3. The flow of in processor scheduling

application. For example, in the task graph of Fig. 2(a), task 1 has the highest level since all the other tasks cannot be executed until task 1 is finished, while task 5 has the lowest level because no following tasks depend on task 5. Detailed description of the algorithm can be found in [5]. After global agent generate the pre-schedule results, like in Fig. 2(b), it will not send this complicated information to each cluster. Instead, global agent will compress the result into a undirected graph called *processor load graph* (PLG) which sketches the work load of each virtual processor. As shown in Fig. 2(c), in the PLG, each node represents a virtual processor of the pre-schedule result and only records the total power consumption and execution time of the virtual processor. The edges are used to denote the amount of messages to be transmitted between different virtual processors. Sending PLG instead of the detailed pre-scheduling result to the clusters significantly reduces the complexity of mapping algorithm and saves large amount of communication between processors.

After each cluster agent receives the PLG from global agent, it will map the virtual processors to the real processors within the cluster. The mapping algorithm is heuristic-based. Virtual processors of the PLG will be mapped one by one according to the descending order of the *importance* value. In selecting the real processor for each virtual processor, the agent will compare the *cost* for mapping the virtual processor on each real processor that has not be assigned yet and the one with lowest cost will be chosen. The heuristics used to compute importance and cost are listed in table I. Using the combination of the three pair of heuristics with different objective will generate the mapping result which could reduce the peak temperature of the system without sacrificing too much computing performance. The most efficient combination of the heuristics we have found yet is to use the three pair of heuristics with the same weight in computing the importance and cost.

After the tasks have been assigned to the processors, processor agents will dynamically decide the execution order of the tasks. In each processor agent, two queues of tasks are maintained: the *wait queue* which contains tasks that are waiting for messages from tasks with precedence and cannot be executed immediately and the *ready queue* which contains tasks that are ready for execution. Whenever the processor finished computing a previous task, the processor agent will be invoked to select a task with highest priority from the ready queue to execute. The priority of the tasks is defined as the level value that is mentioned in pre-scheduling algorithm. This definition of priority guarantees that the execution of an application will not be held back because of the delay of few critical tasks. Fig. 3 shows the scheduling flow for processor agents.

TABLE I
HEURISTICS FOR IMPORTANCE AND COST FUNCTION

Objective	Heuristic for importance	Heuristic for cost
Minimize makespan	Execution start time of the virtual processor	Total execution time remained
Balance power	Power consumption of the virtual processor	Total power remained
Minimize communication cost	Total amount of messages flow in or out of the virtual processor	Total communication delay generated

TABLE II
PEAK TEMPERATURE AND EXECUTION TIME RESULT

NoC Size	Peak temperature ($^{\circ}C$)		Execution time (Kcycle)	
	Distributed	Centralized	Distributed	Centralized
32	99.66	113.59	1.23	1.23
48	95.59	105.89	1.23	1.23
96	92.05	101.64	1.21	1.21

III. EXPERIMENTAL RESULTS

In the experiments, we tested our algorithm on three 2-layer 3D-NoC systems with different processor numbers. The benchmark is composed of 100 randomly generated task graphs. For comparison, we also implemented a centralized scheduling algorithm using the same heuristics as our distributed algorithm. Table II shows the execution time and peak system temperature achieved by the two algorithms. We can see that our algorithm can always achieve lower peak temperature without increasing the execution delay. The scalability of our algorithm also significantly outperforms the centralized algorithm. The complexity of centralized heuristic-based scheduling algorithms relates to the number of processors inside the NoC system linearly. However, the complexity of our distributed scheduling algorithm will not increase with the NoC size since it only depends on the cluster size.

REFERENCES

[1] B. Feero and P. Pande, "Networks-on-Chip in a three-dimensional environment: A performance evaluation," *IEEE Transanctions on Computers*, vol. 58, no. 1, pp. 32–45, Aug. 2009.
[2] W. Hung, G. Link, Y. Xie, N. Vijaykrishan, and M. Irwin, "Interconnect and thermal-aware floorflanning for 3D micro-processors," in *Proceedings of 7th International Symposium on Quality Electronic Design (ISQED'06)*, Mar. 2006, pp. 98–104.
[3] C. Chou and R. Marculescu, "Incremental run-time application mapping for homogeneous NoCs with multiple voltage levels," in *Proceedings of 5th International Coference on Hardware/Sostware Codesign and System Synthesis (CODES+ISSS'07)*, Sept. 2007, pp. 161–1166.
[4] Y. Xie, N. ViJ'aykrishnan, M. Kandemir, and M. Irwin, "Thermal-aware task allocation and scheduling for embedded systems," in *Proceedings of 2005 Design, Automation and Test in Europe (DATE'05)*, Mar. 2005.
[5] Y. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Computing Surveys*, vol. 31, no. 4, pp. 406–471, Dec. 2010.