Journal of Circuits, Systems, and Computers
© World Scientific Publishing Company

THERMAL-AWARE TASK SCHEDULING FOR 3D-NETWORK-ON-CHIP: A BOTTOM TO TOP SCHEME

YINGNAN CUI
School of Computer Engineering, Nanyang Technological University,
N4-02a-32 Nanyang Avenue, Singapore, 639798,
cuiyingnan.hit@gmail.com

WEI ZHANG
Department of Electronic and Computer Engineering,
Hong Kong University of Science and Technology
Clear Water Bay, Kowloon, Hong Kong
wei.zhang@ust.hk

VIVEK CHATURVEDI
School of Computer Engineering, Nanyang Technological University,
N4-02a-32 Nanyang Avenue, Singapore, 639798,
vChaturvedi@ntu.edu.sg

WEICHEN LIU
College of Computer Science, Chongqing University
174 Shazheng Street, Shapingba District, Chongqing, 400044,China
wcliu@cqu.edu.cn

BINGSHENG HE
School of Computer Engineering, Nanyang Technological University,
N4-02a-32 Nanyang Avenue, Singapore, 639798,
bshe@ntu.edu.sg

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

3D-NoC emerges as a potential multi-core architecture delivering high performance, high energy efficiency and great scalability. However, 3D-NoC suffers from severe thermal problems due to its high power density. To solve this problem, thermal-aware scheduling is an effective solution. However, the high complexity of the thermal model of 3D-NoC becomes a major hurdle for developing efficient thermal-aware scheduling algorithms for 3D-NoC. In this paper, we propose a novel thermal aware task scheduling scheme named as the Bottom-to-Top (B2T) approach to address this challenge. This heuristic-based method performs task allocation on processing units to efficiently minimize the peak temperature and improve the execution time of the tasks with low complexity. The algorithm is first designed for 2-layer 3D-NoC and then extended to 3D-NoC with an arbitrary number of layers. When compared to traditional thermal-aware scheduling algorithms designed for 2D-NoC, our B2T algorithm can achieve significant peak temperature reduction (up to 11.9 °C) and performance improvement (up to 4%) on 2-layer 3D-NoC. The improvement becomes more significant as the number of layers in 3D-NoC increases. For 4-layer 3D-NoC, the improvement is up to $13.23°C$ peak temperature reduction.

Keywords: 3D integration; Network-on-Chip; Thermal-aware scheduling

1. Introduction

The development in 3D integration technology of semiconductor devices has brought a new horizon for traditional Network-on-Chips (NoC) systems. When compared to its 2D counterpart, 3D-NoC has the benefits of lower chip area, lower power consumption and lower signal propagation delay.[1] However, the significant increase in power density of 3D-NoC has posed severe challenges to thermal management of the system. As shown in previous studies,[2] the run-time peak temperature of 3D-NoC could be over $20°C$ higher than a 2D-NoC with the same power consumption. Since high temperature could severely damage the performance and the reliability of semiconductor devices, efficient thermal management techniques are crucial for 3D-NoC. In this paper, we investigate thermal-aware task scheduling algorithms that are designed to reduce the peak temperature of 3D-NoC.

Among various thermal management methods for multiprocessor, thermal-aware task scheduling is a light-weight and highly efficient technique. In many studies on thermal-aware scheduling,[3,4,5] the tasks running in the systems are assumed to be independent with each other. The thermal-aware scheduling algorithms proposed by these studies are usually based on the methodology of migrating high-power tasks from hot processors to cool processors in the NoC systems. However, more generally, many applications with high performance requirements are modeled as task graphs.[6] As a result, thermal-aware scheduling algorithms for task graphs have attracted a lot of attention in previous studies.[7,8,9,10]

The task graph scheduling problem is proved to be NP-complete.[11] Some of the thermal-aware task graph scheduling algorithms formulated the problem as a formal optimization problem and used methods like exhaustive search and integer linear programming algorithms to find the optimal solution.[7,8] The high computation complexity of these approaches prohibits their use in run-time cases. Thus, light-weighted approaches are preferred. For example, Hung *et al.* and Stavrou *et al.* proposed heuristic-based solutions which are suitable for on-line scheduling.[9,10] However, all the studies mentioned above are designed for 2D multiprocessors.

Due to the architectural differences between 3D-NoC and 2D-NoC, existing algorithms designed for 2D-NoC are ineffective or infeasible for 3D-NoC. Firstly, the thermal features of 3D-NoC differ significantly from 2D-NoC. The heuristics designed for 2D-NoC usually take the assumption that the temperature of each core in the system is only decided by the power consumption of that core. This is because the heat flow between different cores on a 2D-NoC is negligible compared to the power consumption in each core. However, such an assumption no longer holds on 3D-NoC, where the inter-layer heat flows significantly affect the temperature of each core. Secondly, as the thermal features of 3D-NoC are more complicated than 2D-NoC, the complexity of thermal-aware scheduling algorithms also grows. Some thermal-aware scheduling algorithms estimate the temperature of each core in a 2D-NoC through thermal simulation.[12] Such approaches do not scale as the increase in the complexity of thermal model. It means that using run-time thermal estimation for each core in 3D-NoC is infeasible due to the prohibitive complexity.

To address these issues, it is essential to re-design the heuristics for 3D-NoC for an efficient thermal-aware scheduling algorithm with low complexity. Zhou *et al.* pointed out that the temperature of 3D-NoC is closely related to the power distribution in vertically stacked processors.[13] By placing high-power tasks on layers which are closer to the heat sink, the temperature of 3D-NoC systems could be effectively reduced. In this study, Zhou *et al.* proposed an heuristic-based thermal-aware scheduling algorithm based on their observation. Wang *et al.*[14] proposed a similar scheduling algorithm for 3D-NoC and they combined the scheduling algorithm together with DVFS-based thermal management technologies to achieve better temperature control. However, these algorithms are designed for independent tasks. Cox *et al.* proposed a thermal-aware mapping algorithm for streaming algorithms on 3D multi-processor systems.[15] This algorithm requires the off-line profiling of the thermal characteristics of the 3D multi-processor chips, which makes the algorithm infeasible for run-time uses.

In this study, we propose an efficient thermal-aware task scheduling algorithm for 3D-NoC which is called as the Bottom-to-Top (B2T) algorithm. The algorithm is designed for run-time scheduling of task graphs. The B2T algorithm works in the following manner. First, it schedules all the tasks to the bottom layer of a 3D-NoC to uniformly distribute power consumption among each stack of processors. Second, the algorithm moves low-power tasks to the top layer to reduce the execution time of the applications by exploiting parallelism of the tasks while maintaining low temperature of the cores. Our B2T scheme also takes inter-processor communication of 3D-NoC into consideration to further reduce execution time and power consumption of the system. In this study, we first propose the B2T scheme specifically for 2-layer 3D-NoC, and then extend the generality of the B2T scheme, such that the algorithm could be applied to 3D-NoC with an arbitrary number of layers.

In the experiments, we have compared our B2T scheme with the state-of-the-art thermal-aware scheduling algorithms designed for 2D-NoC,[9,10] since there is no existing work on scheduling task graphs on 3D-NoC. The results show that our B2T algorithm achieves significantly lower peak temperature with better or comparable performance. In 2-layer 3D-NoC, our B2T algorithm shows up to $11.9°C$ peak temperature reduction with comparable execution time when compared thermal-aware scheduling algorithms designed for 2D-NoC. In 3D-NoC with more layers, the advantages of our B2T algorithms become more significant. For 4-layer 3D-NoC, the peak temperature reduction is up to $13.23°C$.

The rest of this paper is organized as follows: Section 2 introduces the thermal model of 3D-NoC and explains the rationale of the B2T scheme. Section 3 presents a formal definition of the problem. Section 4 describes the B2T thermal-aware scheduling scheme for 2-layer 3D-NoC. Section 5 extends the B2T algorithm to 3D-NoC with an arbitrary number of layers. Section 6 shows the experimental results. In the end, Section 7 concludes the paper.

2. Thermal Features of 3D-NoC

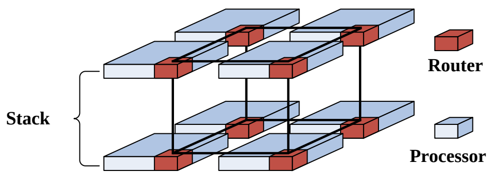

Fig. 1. 2-layer 3D NoC architecture with 3D-mesh topology.

Without loss of generality, we assume all the 3D-NoC systems mentioned in this studied have the 3D-mesh topology. The 3D-mesh technology is directly extended from the mesh topology in 2D-NoC.[16] Fig. 1 uses a minimal example to show the architecture of 3D-NoC. In 3D-NoC, multiple layers of processors with their interconnection networks are stacked together. For ease of discussion, we make the following definitions. In a 3D-NoC, the layer of devices that locates next to the heat sink is called the bottom layer; the layer on the opposite side of the chip is called the top layer; the other layers are called intermediate layers. In addition, in a 3D-NoC, a column of cores that are vertically overlapped with each other is called as a *stack*. The notation of stack is also shown in Fig. 1.

Fig. 2 shows a commonly adopted thermal model for general IC chips.[17] In the thermal model, the thermodynamic characteristics of the chip are illustrated by an equivalent electrical circuit. In the circuit, voltages represent temperatures in the chip. Electrical currents represent heat flows. Current sources represent power consumption. Resistors and capacitors represent the thermal resistance and capacitance of the materials in the heat dissipation routes. During the run-time, the power consumed by an IC chip turns into heat. Some of the heat is absorbed by the material which causes the temperature of the chip to rise. The rest of the heat dissipates into the air through the heat sink. The speed of heat dissipation is defined by Eq. 1. In Eq. 1, Q is the amount of heat dissipated in unit time. ΔT is the temperature difference between the chip and the ambient environment. R is the thermal resistance of the materials in the chip.

$$Q = \frac{\Delta T}{R} \tag{1}$$

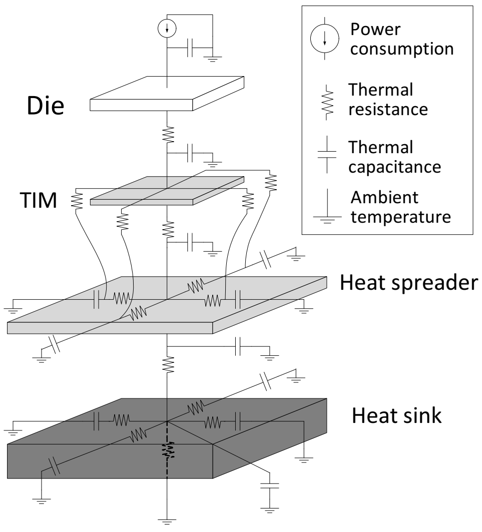

Fig. 2. General thermal mode for IC chips.

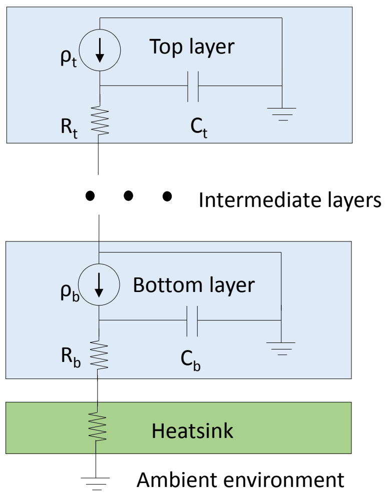

Fig. 3. Simplified thermal model for 3D-IC

Fig. 3 shows a simplified thermal model for 3D-IC which only illustrates the main heat dissipation route. The figure shows that the heat generated at the top layer is much more difficult to dissipate into the environment than the heat generated at the bottom layer. This is because the heat dissipation route of the top layer is longer and therefore the thermal resistance on the route is higher. As a result, we can get the following conclusion. In a 3D-NoC, we assume that the total power consumption is a fixed amount. As the share of the power consumed by the top layer increases, the resulting peak temperature of the chip also rises. This conclusion is also supported by experimental results.[18,19] This observation is the fundamental principle for our thermal-aware scheduling algorithm in this study. In order to reduce the peak temperature of 3D-NoC, the algorithm must keep the power consumption on the top layer lower than the bottom layer.

3. Problem Formulation

In order to give the formal formulation of the thermal-aware scheduling problem for 3D-NoC, we first have to make a few assumptions and definitions.

Firstly, we assume that the 3D-NoC systems mentioned in this paper all have 3D-mesh topology, wormhole flow control and x-y-z routing scheme.[20,21] This is a commonly adopted setting in previous studies. We note that our B2T scheme can be applied to 3D-NoC systems with different kinds of topologies, routing algorithms, and flow control technologies. In addition, we assume that all the cores in the 3D-NoC are homogeneous. The set of cores in one 3D-NoC is denoted by $P = \{p\}$. The subset $P_i \subset P$ denotes the cores located on the i^{th} layer of the 3D-NoC. Specifically, P_b denotes the cores at the bottom layer and P_t denotes the cores on the top layer. The temperature of each core is denoted as $\tau(p)$.

Secondly, we assume that the applications executed by the 3D-NoC are modeled as task graphs. A task graph is denoted as $G(V, E)$, where V is the set of vertices representing the tasks and E is the set of edges representing the precedence relationship and communication between tasks. Each task v in the application has two attributes. Firstly, the the execution time of task v is denoted by $l(v)$. Secondly, the total energy consumed by task v is denoted by $\epsilon(v)$. With $l(v)$ and $\epsilon(v)$, we can compute the average power consumption of task v by Eq. (2), where $\rho(v)$ is the average power consumption.

$$\rho(v) = \frac{\epsilon(v)}{l(v)} \tag{2}$$

To define the schedule of an application, we should determine the following two functions. Firstly, the mapping function, denoted by $M : V \mapsto P$, assigns the individual tasks to the cores for execution. Secondly, the scheduling function, denoted by $S : V \mapsto R_0^+$, defines the time when each task starts execution.

With the above assumptions and definitions, we formulate the thermal-aware scheduling problem as follows:

$$\begin{array}{ll} \text{Given} & G(V, E),\ P, \\ \underset{M,S}{\text{minimize}} & \max\limits_{v \in V}\{S(v) + l(v)\} \quad + \quad \alpha \cdot \max\limits_{p \in P}\{\tau(p)\}. \end{array}$$

In the problem formulation, where $\max_{v \in V}\{S(v) + l(v)\}$ denotes the total execution time of the application, $\max_{p \in P}\{\tau(p)\}$ is the peak temperature of the cores in the 3D-NoC, α is a weighting parameter. In this study, the thermal-aware scheduling problem is a multi-objective optimization problem. The goal of the problem is to minimize the peak temperature and the execution time of the applications at the same time. The parameter α is used to changing the weight of the peak temperature reduction in the optimization goal. When α is zero, the problem becomes a performance optimization problem. We note that minimizing peak temperature without considering the performance of the 3D-NoC is not meaningful. Because we can reduce the peak temperature of the 3D-NoC by unlimitedly slowing down the execution speed of the applications. As a result, the objective function of the problem should always contain the execution time of the application.

The symbols used in this study are summarized in Table 1.

Table 1. Symbol definitions

Symbols	Definition
$l(v)$	The length of the task v, measured in clock cycles.
$\epsilon(v)$	The energy consumed by task v.
$\rho(v)$	The power consumption of task v.
$w(e)$	The size of the message on edge e.
$\Phi(v)$	The priority of task v.
$\tau(p)$	The temperature of processor p.
P_b	Set of processors at the bottom layer of 3D-NoC.
P_t	Set of processors on the top layer of 3D-NoC.

4. Thermal-Aware Task Scheduling for 2-layer 3D-NoC

In this section, we introduce our bottom-to-top thermal-aware scheduling algorithm for the 2-layer 3D-NoC. In Section 5, we extend the algorithm such that it can be applied to 3D-NoC with an arbitrary number of layers.

4.1. *The Bottom-to-Top Scheduling Scheme*

Based on the analysis on the thermal model of 3D-NoC in Section 2, we can conclude that in order to keep the temperature low inside a 3D-NoC, the power consumption on top layer should be lower than the bottom layer. However, this only provides the principle for thermal management in the vertical direction. Horizontally, we first need to balance the power consumption in among stacks in a 3D-NoC to avoid thermal hotspots. If we view a stack of cores as one hyper-core, we could apply the thermal-aware scheduling algorithms designed for 2D-NoC. As shown in Fig. 4, the 3D-NoC with 8 cores on two layers could be viewed as a 2D-NoC with 4 hyper-cores, where each hyper-core contains the processors vertically stacked together. In this way, we transform the 3D-NoC scheduling problem into a 2D-NoC scheduling problem, which can be solved by thermal-aware scheduling algorithms designed for 2D-NoC. After that, we can adjust the power consumption within each stack according to our previous conclusion. We note that assigning tasks to a hyper-core is equivalent to assigning the tasks to the bottom-layer core in the stack. Therefore, we name our algorithm as the bottom-to-top algorithm.

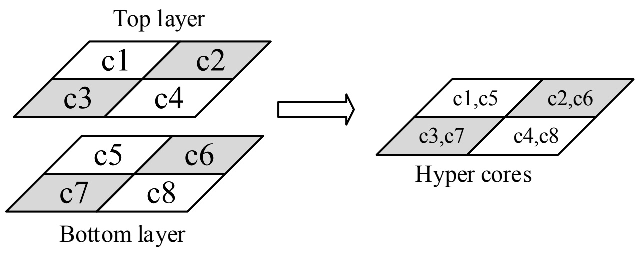

Fig. 4. The example of hyper-core.

Before we introduce the B2T algorithm in detail, we use an example to show the workflow of the algorithm. The example is shown in Fig. 5. The problem is defined by Fig. 5(a) and (b). Fig. 5(a) shows the application waiting for scheduling and the length of each task. Fig. 5(b) shows the target 3D-NoC system which has two layers and four cores. Fig. 5(c) shows results of the first step of the B2T algorithm, where all the tasks are assigned to the cores at the bottom layer. Fig. 5(d) shows the final scheduling result after the second step, where two of the tasks are moved to the top layer. In the rest part of this section, the same example is used for several times to help explaining the algorithm.

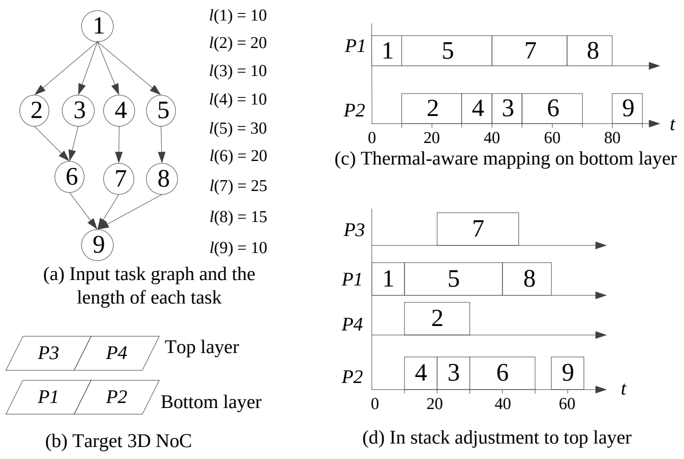

Fig. 5. A example of the two-step thermal-aware task graph scheduling algorithm.

4.2. *Thermal-Aware Task Scheduling at Bottom Layer*

To solve the thermal-aware task scheduling at the bottom layer, we adopt a list-based scheduling algorithm. The basic procedure of list scheduling is as follows. The algorithm keeps a list of tasks that are ready to be executed. For each task in the ready list, we assign a processor to execute the task. After the tasks in the ready list are all scheduled, the algorithm updates the ready list and repeats the above procedures until all the tasks are scheduled.

In a list scheduling algorithm, two factors have major influence on the scheduling results. The first factor is the scheduling order of tasks in the ready list. The second factor is the criteria to select the proper core for each task. To achieve the two objectives of reducing execution time and peak temperature, we use two heuristics in the algorithm. Firstly, for each task in the application, we define a *priority* which reflects the criticality of the task regarding its influence on the execution time of the whole application. The scheduling algorithm always first schedules the task with a higher priority. Secondly, we define a *cost function* to evaluate each core in the system regarding its potentials to affect the performance and the temperature of the system. When assigning cores for each task, the scheduling algorithm always picks the core with the lowest cost. The detailed definitions of priority and cost function are introduced below.

For a given task $G(V, E)$, Eq. (3) defines the priority of each task $v \in V$, which is denoted as $\Phi(v)$. Eq. (3) is a recursive definition, where V_{chd} is the set of children of task v, v' is a child of v, e' is the edge connecting v and v', and $\bar{B}$ is the communication bandwidth of the 3D-NoC. Actually, $\Phi(v)$ stands for the total execution time along the longest path from task v to the end of application. This definition has been used in previous studies about task graph scheduling algorithms.[6]

$$\Phi(v) = \begin{cases} 0 & \text{if } V_{chd}(v) = \emptyset, \\ \max\{\Phi(v') + \bar{B} \times w(e')\} & \text{if } V_{chd}(v) \neq \emptyset. \end{cases} \tag{3}$$

The cost function for mapping a task to a processor is defined in Eq. (4). In the equation, $t_s(v, p)$ is the earliest starting time of task v on processor p, which is determined by the mapping of previous tasks, $Q(v, p)$ is the accumulated heat generated inside the processor, which evaluates the thermal impact of the previously mapped tasks on processor p, and α is the weighting parameter which is used to change the weight of the thermal related term in the cost function. Eq. (4) is a direct reflection of the objective function of the problem defined in Section 3. We note that the value of α should be non-negative. By tuning the value of α, we can change the focus of our thermal-aware scheduling algorithm. In the case where execution time is a more important concern, we can reduce the value of α. And when temperature reduction is a more important concern, we can increase the value of α. The influence of the weighting parameter α is shown in the experiments in Section 6.

$$c(p, v) = t_s(v, p) + \alpha \times Q(v, p) \tag{4}$$

The definition of $Q(v, p)$ is further expanded in Eq. (5), where v^* represents the task that has already been mapped to processor p, and q_{out} is an estimation of heat dissipated from the processor to the environment in a time unit. The first two terms of Eq. (5) represent the total heat generated inside the processor while the last term evaluates the total amount of heat dissipation of the processor before the task finishes.

$$Q(v, p) = \sum_{v^*} \epsilon(v^*) + \epsilon(v) - q_{out} \times (t_s(v, p) + l(v)) \tag{5}$$

4.3. *In-Stack Adjustment to Top Layer*

After the scheduling algorithm at the bottom layer finishes, the in-stack adjustment algorithm is performed. The algorithm exploits the parallelism of the tasks and selectively picks some of the tasks in each stack and moves them to the top-layer processors in order to further reduce execution time while maintaining low temperature. Algorithm 1 shows the pseudo code of the in-stack adjustment algorithm.

The algorithm first makes a list of tasks which have been assigned to the stack by the previous step (Line 2), and then the algorithm sorts these tasks according to the descending order of the priorities. These tasks are denoted by V_s. In order to identify the tasks that can be executed in parallel as mentioned above, the algorithm runs a function to divide the tasks into groups which contain tasks that can be executed simultaneously (Line 3). Such groups are called *parallel groups* and are denoted by V_{pg}. The union of all the parallel groups is denoted as U_{pg}.

The formation of parallel groups is performed as follows. Firstly, we pick a task with maximum priority from V_s as the head task of the parallel group, v_{head}. Secondly, insert all tasks from V_s which fulfill the requirement in Eq. (6) into the parallel group. Eq. (6) guarantees that all the tasks inside V_{pg} do not have direct precedence relationship with each other. It means that the tasks in V_{pg} are able to be executed in parallel. By repeating the two steps, we can divide V_s into multiple parallel groups.

$$\textbf{If}\ \ \Phi(v_{head}) - l(v_{head}) > \Phi(v), \ \ \textbf{then}\ \ v \in V_{pg}. \ \ (v \in V_s) \tag{6}$$

Because tasks of one parallel group can be executed in parallel, we can then move some of the tasks inside the parallel group to the processor on the top layer to reduce the execution time. This job is done by Line 7 to Line 17 in the algorithm. We note that only the parallel groups which contain more than one task can be adjusted (Lines 8 to 16). For those parallel groups with a single task, we merge them into a special set of tasks, denoted by V_{ndef}, for later consideration (Line 15).

Algorithm 1 In-stack adjustment algorithm

```
    Input: $G(V,E)$, $M_b$, $P_b$, $P_t$, $\Phi$
    Output: $M$
 1: for all $p_b \in P_b$ do
 2:     $V_s$=find_tasks_mapped_in_stack($p$, $M_b$);
 3:     $V_s$=sort_tasks_by_priority($V_s$);
 4:     $U_{pg}$=divide_parallel_groups($V_s$);
 5:     $p_t$=find_top_layer_processor($p$);
 6:     $V_{ndef} = \emptyset$;
 7:     for all $V_{pg} \in U_{pg}$ do
 8:         if $number\_of\_tasks(V_{pg}) > 1$ then
 9:             $V_t$=adjust_tasks_to_top_layer($V_{pg}$);
10:             $V_b = V_s \cap \bar{V_t}$;
11:             for all $v \in V_{pg}$ do
12:                 $M(v) = (v \in V_b)\ ?\ p_b : p_t$;
13:             end for
14:         else
15:             $V_{ndef}$= $V_{ndef} \cup \{v\}$ ;
16:         end if
17:     end for
18:     for all $v \in V_{ndef}$ do
19:         $o_b$=compute_overhead($v$, $p_b$);
20:         $o_t$=compute_overhead($v$, $p_t$);
21:         $M(v) = (o_b - o_t < o_{th})\ ?\ p_b : p_t$;
22:     end for
23: end for
```

Line 9 selects the tasks to be moved to the processor on the top layer, and the rest tasks of V_s remain at the processor at the bottom layer (Line 10). The tasks of V_t and V_b must fulfill two conditions. Firstly, for any $v_t \in V_t$ and $v_b \in V_b$, there is $\rho(v_t) < \rho(v_b)$. This condition guarantees that the power consumption of tasks mapped to the top layer are lower than tasks mapped to the bottom layer. It helps maintain low temperature of the system, as discussed in Section 2. Secondly, the total length of the tasks mapped to the top layer and should be shorter than the tasks mapped to the bottom layer ($\sum l(v_t) < \sum l(v_b)$). It prevents temperature increases caused by load imbalance between the processors in one stack.

Tasks from parallel groups with a single task are handled by Lines 18 to 22 of Algorithm 1. The mapping of these tasks depends on the communication overhead between tasks as defined in Eq. (7). The Eq. computes the total amount of messages on the edges which connect task v to other tasks that are previously mapped to the processor on the other layer. As shown in Line 21, only when the communication overhead of mapping task v to the bottom layer is too high, v could be mapped to the top layer. Tasks in the single-task parallel groups cannot start execution until all the previous tasks be finished. Therefore, in most cases, only communication delay can affect the start time of these tasks. For the sake of temperature reduction, we keep these tasks in bottom layer, except for the cases when the communication overheads become overwhelming. The worst-case complexity of in-stack adjustment algorithm is $O(V)$.

$$\begin{cases} o_b = \sum_{v_t} w(e_t), e_t = v_t \rightarrow v, \text{or } v \rightarrow v_t, \\ o_t = \sum_{v_b} w(e_b), e_b = v_b \rightarrow v, \text{or } v \rightarrow v_b. \end{cases} \tag{7}$$

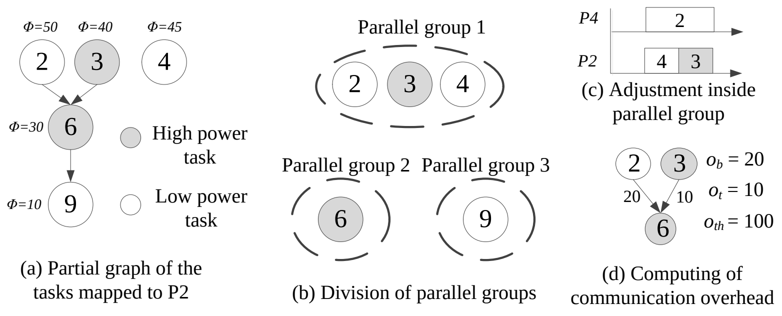
Φ=50
Φ=40
Φ=45
2
3
4
Φ=30
6
High power task
Φ=10
9
Low power task
(a) Partial graph of the tasks mapped to P2
Parallel group 1
2
3
4
Parallel group 2
Parallel group 3
6
9
(b) Division of parallel groups
P4
2
P2
4
3
(c) Adjustment inside parallel group
2
3
20
10
6
o_b = 20
o_t = 10
o_th = 100
(d) Computing of communication overhead

Fig. 6. In-stack adjustment algorithm example. (Inside the stack of processor P2 and P4 as shown in Fig. 5(b))

Fig. 6 illustrates the details of in-stack adjustment algorithm for previous example shown in Fig. 5. Fig. 6(a) shows the partial task graph containing the tasks that have been assigned to core P2 in the previous example. In the partial task graph, the high-power tasks are denoted by the grey nodes and the low-power tasks are denoted by the white nodes. Fig. 6(b) shows how the tasks are divided into the parallel groups according to Eq. (6). Fig. 6(c) shows the adjustment result of the parallel group 1 in the example. In the result, low-power task 2 is adjusted to the top layer. Fig. 6(d) shows the results of computing communication overhead for the tasks in single task parallel groups. In the example, to decide the final location of task 6, which is the only task in parallel group 2, the algorithm has to compute the communication overhead o_b and o_t according to Eq. (7). Since the overhead of mapping task 6 to bottom layer is not very overwhelming, we keep task 6 in the bottom layer. The final result of the in-stack adjusting algorithm is already shown in Fig. 5(d) in Section 4.1.

5. Extension to Multi-Layer 3D-NoC

When extending the B2T algorithm to 3D-NoC with an arbitrary number of layers, we do not need to change the first step of the algorithm. Major extension needs to be applied to the in-stack adjustment algorithm to meet the requirements for 3D-NoC with more than two layers.

Based on Algorithm 1, the in-stack adjustment algorithm can be adjusted with a recursive manner. Assume that a 3D-NoC has N layers. In the extended in-stack adjustment algorithm, we first divide the N cores in each stack into two subsets, each of which contains $N/2$ cores. We view the two subsets of cores as two hyper-cores as mentioned in Section 4.2. Next, we can apply the Algorithm 1 to assign the tasks to the two hyper-cores. Within each hyper-core, we can further divide the cores into two hyper-cores and recursively repeat the procedure until there exists no hyper-core containing more than one core. We note that sometimes the hyper-core may contain odd numbers of cores. In such cases, when dividing the cores, we have to make sure the hyper-core located closer to the bottom layer contains one less core than the other. In this way, we can guarantee that the power consumption of the cores closer to the heat sink always consume more power than cores located further away from the heat sink. The pseudo code of the extended in-stack adjustment algorithm is shown in Algorithm 2.

Algorithm 2 In-stack adjustment algorithm with multi-layer extension

```
    Input: $M_b$, $P_s = \{p_i,\ p_{i+1},\ \dots\ ,\ p_j\}$
    Output: $M$
 1: $M = \emptyset$
 2: **if** $i == j$ **then**
 3:     $M+ = M_b$;
 4:     return;
 5: **else if** $i + 1 = j$ **then**
 6:     $(M_i, M_j)$ = in_stack_adjust_2layer($M_b$, $p_i$, $p_j$); //Algorithm 1.
 7:     $M = M + M_i + M_j$;
 8:     return;
 9: **else**
10:     $k = (i + j)/2$;
11:     $P_b = p_i,\ \dots\ , p_{k-1}$;
12:     $P_t = p_k,\ \dots\ , p_j$;
13:     $(M_b, M_t)$ = in_stack_adjust_2layer($M_b$, $P_b$, $P_t$);
14:     in_stack_adjust($M_b$,$P_b$);
15:     in_stack_adjust($M_t$,$P_t$);
16: **end if**
```

Fig. 7 uses an example to show how the extended in-stack adjustment algorithm works on a 4-layer 3D-NoC. In the example, the four cores in the stack are divided into two hyper-cores and the tasks are adjusted into the two hyper-cores according Algorithm 1. Next, within each hyper-core of processors, Algorithm 1 is invoked again to assign tasks to each individual processor.

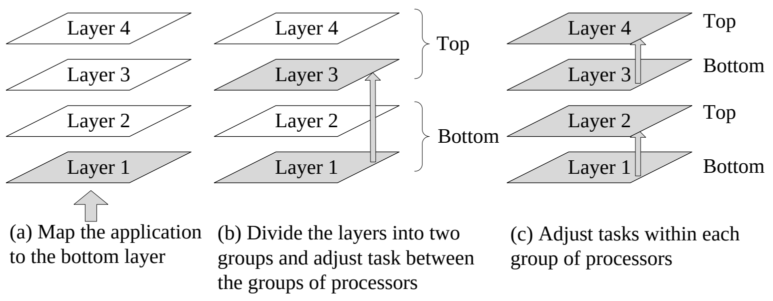
Layer 4
Layer 3
Layer 2
Layer 1
(a) Map the application to the bottom layer
Layer 4
Layer 3
Layer 2
Layer 1
Top
Bottom
(b) Divide the layers into two groups and adjust task between the groups of processors
Layer 4
Layer 3
Layer 2
Layer 1
Top
Bottom
Top
Bottom
(c) Adjust tasks within each group of processors

Fig. 7. Apply Bottom-to-Top algorithm in a four-layer 3D-NoC

6. Evaluation

In this section, we evaluates the performance of our B2T algorithm on a 2-layer 3D-NoC and a 4-layer 3D-NoC. The results are compared to state-of-the-art thermal-aware scheduling algorithms designed for 2D-NoC.

6.1. *Experimental Results for 2-layer 3D-NoC*

Setup. The experiments are carried out on a target 3D-NoC system which has eight processors equally placed on two layers. The 8-core processor provides enough computing resources to show the effectiveness of the scheduling algorithms. HotSpot 5.1 was used as the thermal simulation tool.[17] The thermal parameters of HotSpot are shown in Table 2. The other parameters follow the default settings of HotSpot. The similar floor plan setting could be found in previous studies.[14].

Table 2. Experiments of real application based task graphs

Parameter	Value
Die thickness	$0.15mm$
Inter-layer material thickness	$0.02mm$
Die capacitance	$1.76 \times 10^{-6}\ J/(m^3 \cdot K)$
Die resistance	$0.01mK/W$
Inter-layer material resistance	$0.25mK/W$

In the experiments, we use two kinds of benchmarks. The first group of benchmarks are randomly generated task graphs and the second group of benchmarks are real-application-based task graphs. The randomly generated task graphs are selected from Standard Task Graph (STG).[22] The STG task graph set is commonly adopted in previous studies to evaluate the performance of scheduling algorithms.[23,24] In the experiments, five sets of randomly generated task graphs are used; each set contains 100 task graphs; the number of nodes inside each task graph are 50, 100, 300, 500, 750 respectively. The real application based task graphs is adopted from.[25]

For comparison, we implement two efficient heuristic-based thermal-aware scheduling algorithms designed for 2D-NoC in the experiments. The first algorithm is the heuristic-based algorithm proposed by Hung *et al.*[9] which aims at minimizing the average power consumption in each core in a NoC (Min. Pow.). The other comparison algorithm is the heuristic proposed by Stavrou *et al.* [10] which maps the task with the highest power consumption to the coolest core in the NoC (coolest). In the experiments, the weighting parameter α of the cost function defined in Eq. (4) is set to 1.

In the experiment, we first run the scheduling algorithms to produce the schedule of tasks. Then the power trace of accumulated power consumption of all processors during the task execution is generated according to the schedule. Finally, HotSpot simulates the transient temperature of the 3D-NoC corresponding to the power trace.

Results. In the experiments, since the peak temperature and execution time of each application largely depends on the input task graph, it is infeasible to directly compare the peak temperature and execution time of each application. Therefore, we define two new metrics for the evaluation. The first metric is the *average relative peak temperature*, denoted by $\bar{\Delta\tau}$. $\bar{\Delta\tau}$ is defined by Eq. (8), where τ_i is the peak temperature of the i^{th} application in one benchmark set. The second metric is the *normalized average execution time*, denoted by $\bar{T}$. Eq. (9) defines $\bar{T}$, where t_i is the execution time of the i^{th} application in the benchmark set. Using $\bar{\Delta\tau}$ and $\bar{T}$ as the metrics in the experiments could effectively eliminates the influence caused by different input task graphs.

$$\bar{\Delta\tau} = \text{avg}\{\tau_i - \min_i(\tau_i)\} \tag{8}$$

$$\bar{T} = \text{avg}\{\frac{t_i}{\min_i(t_i)}\} \tag{9}$$

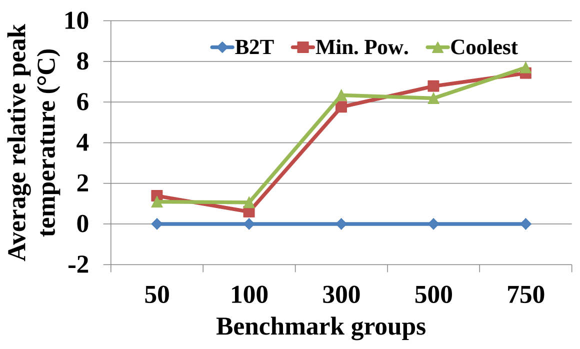

Fig. 8. Comparison of the relative peak temperature.

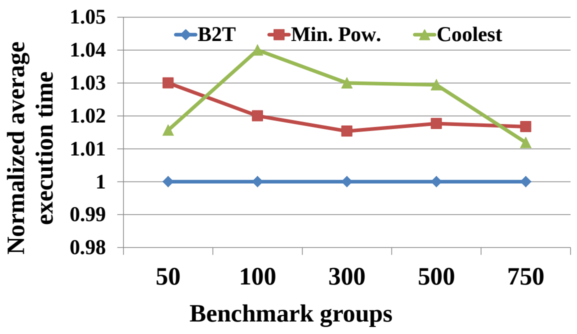

Fig. 9. Comparison of average normalized execution time.

Fig. 8 compares the average relative peak temperature of the five benchmark groups achieved by the three algorithms. From the figure we can see that our B2T scheduling algorithm achieves significant peak temperature reduction compared to the other two algorithms. The average reduction is $4.38°C$ and the maximum reduction is $7.69°C$. This is because our algorithm keeps low-power tasks on the top layer processors and high-power tasks on the bottom layer processors. Such power distribution is thermal-efficient according to the observation mentioned in Section 2. Fig. 9 compares the normalized average execution time of random task graphs achieved by the three algorithms. The average execution time are all normalized to the largest average execution time collected in the experiments. It shows that our algorithm outperforms the other two algorithms by 2% on average with a maximum performance saving of 4%.

Table 3. Experiments of real application based task graphs

Benchmark	Robot		Sparse		fpppp	
Algorithm	Time [a]	Temp.[b]	Time	Temp.	Time	Temp.
B2T	1264	110.75	596	102.84	1350	125.04
Min. Pow.	1236	117.25	568	114.73	1284	134.63
Coolest	1218	119.79	608	113.54	1294	130.77

[a]The execution time is measured in 10^2 clock cycles.
[b]The peak temperature of the 3D-NoC is measured in °C.

To further verify the performance of the B2T scheme, we carry out the experiments on real-application-based task graphs. As mentioned above, three task graphs, Fpppp, Sparse and Robot are selected from.[25] Table 3 compares the execution time of the applications and peak temperature of the 3D-NoC achieved by the three algorithms. The B2T algorithm shows significant temperature reduction in all cases. For each of the three applications, up to $8.96°C$, $11.89°C$ and $9.59°C$ peak temperature reduction are achieved respectively. Although the execution time achieved by our B2T scheme is not always the minimum, it is comparable to the execution time achieved by the other two algorithms. The maximum increase in execution time is within 5%.

6.2. *Experimental Results for 4-layer 3D-NoC*

Table 4. Experiments of real application based task graphs on 4-layer 3D-NoC

Benchmark	Robot		Sparse		Fpppp	
Algorithm	Time [a]	Temp.[b]	Time	Temp.	Time	Temp.
B2T	1318	130.19	566	121.19	1417	147.32
Min. Pow.	1297	139.31	583	132.50	1393	156.23
Coolest	1370	142.69	625	134.42	1435	152.50

[a]The execution time is measured in 10^2 clock cycles.
[b]The peak temperature of the 3D-NoC is measured in °C.

To evaluate the performance of the extended B2T algorithm for 3D-NoC with more than 2-layers, we carry out the experiments on a 4-layer 3D-NoC. In order to fairly compare the difference between the results of 2-layer and 4-layer 3D-NoC. The 4-layer 3D-NoC also contains eight cores. The cores align in a $1 \times 2 \times 4$ array. We test the execution time and peak temperature of the real-application-based benchmarks in the experiment and the results are shown in Table 4. Compared to the results shown in Table 3, the peak temperature of the 4-layer 3D-NoC is significantly higher due to the increase in power density. Same as the experiments for the 2-layer 3D-NoC, the B2T algorithm also shows significant temperature reduction in the 4-layer 3D-NoC. For each of the three applications, up to $12.50°C$, $13.23°C$ and $8.91°C$ peak temperature reduction are achieved respectively. The differences in execution time achieved by the B2T algorithm and the other two algorithms are also always within 5%. The results show that the extended B2T algorithm is very efficient in reducing the peak temperature for 3D-NoC with more than two layers.

6.3. *The Influence of the Weighting Parameter*

As discussed in Section 4.2, the weighting parameter α in the cost function Eq. (4) could affect the result of the experiments. Table 5 shows the execution time and peak temperature of the Fpppp application in the 2-layer 3D-NoC achieved by our B2T algorithm with different α settings. When α equals to 0, the algorithm aims at minimizing the execution time of the application. The resulting peak temperature is also the highest since the thermal concern is removed. As α increases, the execution time of the application also increases and the peak temperature decreases. However, when α is larger than 2, the trend of execution time increasing and peak temperature decreasing slows down. This is because when α is large enough the performance concern becomes negligible and the scheduling problem can be viewed a single-

Table 5. The execution time and peak temperature of Fpppp with different α values.

α	Execution time (10^2 cylces)	Peak temperature (°C)
0	1273	131.24
0.5	1287	128.36
1	1350	125.04
2	1436	118.54
3	1441	118.03

objective optimization problem. In this condition, further increasing α could not make significant changes.

7. Conclusion

Thermal management has been an important issue for 3D-NoC design. In this paper, we proposed an efficient thermal-aware task graph scheduling algorithm called B2T (Bottom-to-Top) scheme. The scheme first maps the task graph at the bottom layer of the 3D-NoC and then selectively adjusts some task to the top layer to reduce execution time while maintaining low operating temperature. We further extend the algorithm for 3D-NoC with an arbitrary number of layers. Experimental results on 2-layer 3D-NoC shows up to 11.9° reduction in peak temperature and up to 4% execution time reduction when compared to previous task graph based method. In the experiments with 4-layer 3D-NoC, our B2T algorithm achieves greater temperature reduction (up to $13.23°C$).

Acknowledgements

A preliminary version of this paper has been published in a conference paper[26]. This work was supported by MoE AcRF Tier 2 grant (MOE2012-T2-1-126) of Singapore. We thank anonymous reviewers and Hui Yan Cheah for their comments on the paper.

References

1. A. B. Ahmed and B. A. Abderazek, Low-overhead routing algorithm for 3D Network-on-Chip, *Proc. Int. Conf. on Computing, Networking and Communications*, Maui, HI, Januray 2012, pp. 23–32.
2. B. Black and *et al.*, Die stacking (3D) microarchitecture, *Proc. 39th Annual IEEE/ACM Int. Symp. on Microarchitecture*, Orlando, FL, December 2006, pp. 469–479
3. A. Kumar, L. Shang, L. Peh, and N. Jha, HybDTM: A coordinated hardware-software approach for dynamic thermal management, *Proc. 43rd annual Design Automation Conf.*, San Francisco, CA,July 2006, pp. 548–553.
4. J. Choi, C.Y.Cher, H. Franke, H. Hamann, A. Weger, and P. Bose, Thermal-aware task scheduling at the system software level, *Proc. Int. Symp. on Low Power Electronics and Design*, Portland, OR, August 2007, pp. 213–218.

5. J. Hu and R. Marculescu, Energy-aware communication and task scheduling for Network-on-Chip architectures under real-time constraints, *Proc. Design, Automation and Test in Europe Conference and Exhibition*, Paris, France, Feburary 2004, pp. 234–239.
6. Y. Kwok and I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Comput. Surv.* **31** (2010) 406–471.
7. M. Chrobak, C. Dürr, M. Hurand, and J. Robert, Algorithms for temperature-aware task scheduling in microprocessor systems, *Algorithmic Aspects in Information and Management* (2008) 120–130.
8. S. Zhang and K. S. Chatha, Approximation algorithm for the temperature-aware scheduling problem, *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, San Jose, CA, Nov 2007, pp. 281–288.
9. Y. Xie, N. ViJ'aykrishnan, M. Kandemir, and M. Irwin, Thermal-aware task allocation and scheduling for embedded systems, *Proc. Design, Automation and Test in Europe Conference and Exhibition*, Messe Munich, Germany, March 2005, pp. 898–899.
10. K. Stavrou and P. Trancoso, Thermal-aware scheduling for future chip multiprocessors, *EURASIP J. Emb. Sys. 2007* **1** (2007) 40–40
11. E.G. Coffman, J.L. Bruno, Computer and job-shop scheduling theory. John Wiley & Sons, 1976.
12. A. Coskun, T. Rosing, K. Whisnant, and K. Gross, Temperature-aware MPSoC scheduling for reducing hot spots and gradients, *Proc. 13th Asia and South Pacific Design Automation Conf.*, Seoul, Koreal, Janurary 2008, pp. 49 –54.
13. X. Zhou and *et al.*, Thermal management for 3D processors via task scheduling, *Proc. 37th Int. Conf. on Parallel Processing*, Portland, OR, September 2008, pp. 115 – 122.
14. H. Wang, Y. Fu, and T. Liu, Thermal management via task scheduling for 3D NoC based multi-processor, *Proc. 2010 Int. SoC Design Conf.*, Jeju, Korea, November 2010, pp. 440–444.
15. M. Cox, A.K. Singh, A. Kumar and H. Corporaal, Thermal-aware mapping of streaming applications on 3D Multi-Processor Systems, *Proc. IEEE 11th Symp. on Embedded Systems for Real-time Multimedia*, Montreal, QC, October 2013, pp. 11–20.
16. l. Benini and G. D. Micheli, Networks on chips: A new SoC paradigm, *Computer*, **35** (2002) 70–78.
17. K. Skadron and *et al.*, Temperature-aware microarchitecture, *ACM SIGARCH Computer Architecture News* **31** (2003) 2–13.
18. B. Black and *et al.*, Die stacking 3D microarchitecture, *Proc. 39th Annual IEEE/ACM Int. Symp. on Microarchitecture*, Orlando, FL, December 2006, pp. 469–479.
19. K. Puttaswamy and G. Loh, Thermal analysis of a 3D die-stacked high-performance microprocessor, *Proc. 16th ACM Great Lakes symp. on VLSI*, Philadelphia, PA, April 2006, pp. 19–24.
20. L. M. Ni and P. K. McKinley, A survey of wormhole routing techniques in direct networks, *Computer* **26** (1993) 62–76.
21. F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, Design and management of 3d chip multiprocessors using network-in-memory, *ACM SIGARCH Computer Architecture News 34* (2006) 130–141.
22. T. Tobita and H. Kasahara, A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *J. of Scheduling* **5** (2002) 379–394.
23. M. Aggarwal, R. D. Kent, and A. Ngom, Genetic algorithm based scheduler for computational grids. *Proc. 19th Int. Symp. on High Performance Computing Systems and Applications*, Guelph, ON, May 2005, pp. 209–215.
24. L. Yang and *et al.*, A framework for partitioning and execution of data stream appli-

cations in mobile cloud computing. *ACM Sigmetrics Performance Evaluation Review* **40.4** (2013) 23–32.
25. W. Liu and *et al.*, A noc traffic suite based on real applications, *Proc. IEEE Computer Society Annual Symp. on VLSI*, Chennai, Inida, July 2011, pp. 66–71.
26. Y. Cui, W. Zhang, V. Chaturvedi, W. Liu, and B. He, Thermal-aware task scheduling for 3D-NoC: a bottom-to-top scheme, *Proc. Int. Symp. on Integrated Circuits*, Singapore, December 2014, pp. 224–227.