On Runtime Communication- and Thermal-aware Application Mapping and Defragmentation in 3D NoC Systems

Bing Li, Xiaohang Wang, *Member, IEEE,* Amit Kumar Singh, *Member, IEEE,* Terrence Mak, *Senior Member, IEEE,*

Abstract—Many-core systems connected by 3D Networks-on-Chip (NoC) are emerging as a promising computation engine for systems like cloud computing servers, big data systems, *etc.* Mapping applications at runtime to 3D NoCs is the key to maintain high throughput of the overall chip under a thermal/power constraint. However, the goals of optimizing both the communication latency and chip peak temperature are contradicting due to several reasons. Firstly, exploiting the vertical TSV links can accelerate communications, while low peak temperature prefers that the tasks to be mapped closer to the heat sink, instead of using the vertical links. Secondly, mapping tasks in close proximity can reduce communication latency, but at the cost of poor heat dissipation. To address these issues, in this paper, we propose an efficient runtime mapping algorithm to reduce both communication latency and overall application running time under thermal constraint. In essence, this algorithm first selects a 3D cuboid core region of a specific shape for each incoming application by setting the region's number of occupied vertical layers and its distance to the heat sink, in order to optimize its communication performance and peak temperature. Next, the exact locations of the core regions in the chip are determined, followed by a task-to-core mapping. A defragmentation algorithm is also proposed to keep free core regions contiguous. The experimental results have confirmed that, compared to two recently proposed runtime mapping algorithms, our proposed approach can reduce the total running time by up to 48% and communication cost by up to 44%, with a low runtime overhead.

Index Terms—3D NoC, Application-based mapping, Thermal management, Run-time, Defragmentation

1 Introduction

Many-core systems have been widely used as an engine to provide sufficient computation power in cloud computing servers, big data systems, *etc*. In these systems, multiple applications with various workload characteristics arrive and leave the system at runtime. Mapping tasks to cores online is the key to improve system performance.

Three dimensional (3D) integration can improve the system integration and reduce global wire length. Through-Silicon-Via (TSV) is one of the popular approaches among various 3D integration techniques [1, 2]. 3D networks-on-chip (NoC) adopting the TSV technique have lower network latency and power consumption, and higher bandwidth [3, 4]. However, as more dies stacked vertically, power density (W/m^2) increases, and the length of heat conduction path increases, resulting in higher propagation delay and higher leakage power [5].

The major challenge of runtime application mapping in 3D NoC is to achieve the contradicting goals of optimizing communication and temperature, which requires to address the following two aspects:

1) Whether to exploit the vertical links or not leads to different communication and temperature behaviors. Lower communication latency requires that the tasks of an application to exploit the TSV connections as much as possible. That is, the tasks with high communication volume should be mapped across multiple vertically adjacent layers in the

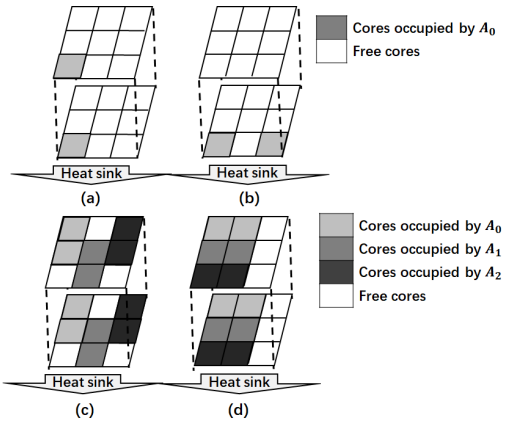

Fig. 1: (a) Mapping tasks to exploit vertical wires for higher communication efficiency. (b) Mapping tasks close to heat sink for lower temperature. (c) Free cores scattered. (d) Free cores packed in close proximity.

- *Bing Li and Xiaohang Wang are with the School of Software Engineering, South China University of Technology, Guangzhou, China, 510006.*
- *Amit Kumar Singh is with the School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, United Kingdom. E-mail: a.k.singh@essex.ac.uk .*
- *Terrence Mak is with the School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom, and with the Guangzhou Institute of Advanced Technology, CAS, Guangzhou, 511458, China. E-mail: tmak@ecs.soton.ac.uk .*

Manuscript received ; revised .

same column, as TSVs provide high bandwidth and shorter distance as in Fig. 1 (a). On the other hand, lower temperature requires that the tasks to be mapped to the layer close to the heat sink as in Fig. 1 (b), instead of across multiple layers.

2) For a system with several running applications, whether to map tasks in a scattered manner or close proximity might lead to fragmentation or high temperature. A phenomenon called "fragmentation" often occurs, causing free cores scattered (not forming a contiguous region). Frequent application arrival and departure lead to fragmentation, *i.e.*, free cores are scattered. As a result, incoming applications might be either mapped to two disconnected core regions with uncontiguous mapping, or wait until a contiguous free core region is formed with contiguous mapping. In the former case, communication latency is increased, and in the latter case, the waiting time is increased, both cause the system throughput to degrade. Fragmentation increases communication distance and latency for tasks of incoming applications as they are mapped to noncontiguous free cores regions [6], as in Fig. 1 (c). Mapping tasks in close proximity alleviates fragmentations. However, if thermal effect is taken into consideration, mapping tasks in close proximity accumulates heat faster and tends to have high peak temperature as in Fig. 1 (d), where the peak temperature is 2°C higher than that in Fig. 1 (c)[1].

In this paper, we propose an algorithm to address the above challenges, in order to optimize the communication and computation performances under a thermal constraint in 3D NoC systems. The algorithm has three steps. First, a 3D cuboid core region of a specific shape is selected for each application. Second, the exact locations of the core regions in the chip are determined, followed by a task-to-core mapping. Finally, during task running, the defragmentation process is executed.

Our previous work [7] has been significantly extended by making the following new contributions:

1) Related work is extended to cover the defragmentation and other mapping algorithms.
2) A thermal-aware defragmentation algorithm is proposed to reduce system fragmentation and waiting time (by bringing free cores into a near-convex contiguous region) for incoming application under thermal constraint.
3) Experiments are substantially extended, including the new evaluation of the proposed defragmentation algorithm and fragmentation metric validation.

The rest of paper is organized as follows. Related work is reviewed in Section 2. The problem is formulated in Section 3. The proposed runtime mapping and defragmentation algorithms are detailed in Section 4. Experimental results are evaluated in Section 5. Finally, Section 6 concludes the paper.

1. The experimental setup is described in Section 5.1

2 Related Work

Existing runtime application mapping algorithms in NoC can be classified into 3 categories: 1) communication oriented, 2) temperature oriented, and 3) both communication and temperature oriented.

Mapping algorithms in the first category focus on communication optimization, improving system throughput by reducing network latency [8, 9]. Most of the algorithms in this category target 2D NoC systems as in [8, 10, 11]. These approaches map communicating tasks to cores close to each other so that communication latency and power are reduced. For example, in [11], Fattah *et al*. proposed a stochastic hill climbing algorithm that starts from a first node and maps the tasks to a set of nodes forming a contiguous region around it. A few mapping algorithms target 3D NoC system as in [5, 9]. In [9], Ziaeeziabari *et al*. proposed a latency-aware task mapping algorithm. It divides communications of a given application's task graph into low volume and high volume communication sub-task-graphs. Then it maps these sub-task-graphs one by one based on their total communications considering where the vertical channels are located in the network. However, these approaches do not consider thermal aspects.

Mapping algorithms in the second category focus on temperature optimization [12–16]. In [12], Cui *et al*. proposed the B2T algorithm that maps all the tasks to the bottom layer which is close to the heat sink, followed by a second step which moves low-power tasks to the top layer. In [15], Zhu *et al*. proposed the temperature-aware partitioning and placement (TAPP) algorithm to avoid thermal hotspots. TAPP spreads high-power cores and routers across the chip by performing a hierarchical bi-partitioning of the cores and concurrently conducting the placement of the cores onto tiles, which achieves both high efficiency and scalability. These algorithms ignore communication distance, which might lead to a higher network latency. In [16], Hamedani *et al*. considered the temperature constraints for thermal-aware mapping of 3D networks-on-chip, focusing on the design of thermal management algorithm.

Mapping algorithms in the third category focus on jointly optimizing communication latency and temperature [17–22]. In [17], Mosayyebzadeh *et al*. proposed an algorithm using fuzzy logic to adjust the impact of heat emission capability, inter-task distance inside application, and distance from hotspots. It reduces the communication delay by mapping tasks with large communication volumes to cores close to each other. It also reduces power consumption and peak temperature by mapping tasks to cores that are close to the heat sink. However, this method does not fully exploit the vertical links to optimize latency.

All the above application mapping algorithms might lead to a phenomenon called *fragmentation* [6], where free cores are scattered. As a consequence, tasks of incoming applications have to be mapped to these scattered cores, leading to increased communication cost. To alleviate fragmentation, a few task migration based approaches were proposed [6, 23, 24]. Moraes *et al*. [23] evaluated the cost of the task migration, demonstrating that the cost to migrate a given task has a small impact on the system performance, enabling it use to improve the overall system performance.

TABLE 1: Symbol Definition

Symbol	Definition
$G(C, L)$	A 3D mesh NoC system
C	The set of cores in a 3D NoC system
L	The set of links connecting the cores in the 3D NoC system
G_w	X dimension (width) of the 3D NoC
G_l	Y dimension (length) of the 3D NoC
G_h	Z dimension (height) of the 3D NoC
S	The set of applications
A_i	Application i
T_i	The set of tasks of A_i
E_i	The set of edges in the task graph of A_i
$e(i, j)$	Edge e between two tasks i and j
$M(t) = c$	A mapping function binding task t to core c
$Tr(e)$	The transmission time of e between two communicating tasks
$V(i, j)$	The traffic volume between two communicating tasks i and j
$D(i, j)$	The Manhattan distance between two cores i and j
$P(c)$	The power of the core c
$P_M(c)$	thermal power capacity of the core c
$\sigma(S)$	The overall running time of applications in the set S
V_i	The average communication volume of each task in A_i
RP	The reference point
CI_i	The corner index for A_i
NOL_i	The number of occupied layers of an application
MD_i	The minimal distance to heat sink of an application
RT_i	The running time of an application
ERT_i	The estimated running time of an application
$\hat{\sigma}(S)$	The estimated overall running time of a set of applications
$\hat{\sigma}^*(S)$	The minimum overall running time of a set of applications
F	The fragmentation metric

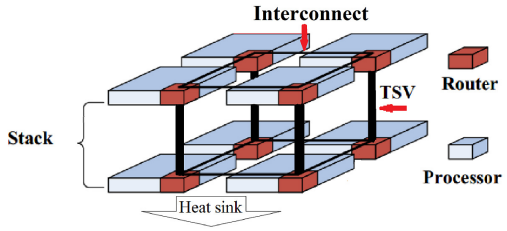

Fig. 2: An example of 3D mesh NoC architecture.

In [6], Ng. *et al*. proposed a defragmentation algorithm to migrate isolated free cores toward one of the chip's corner to cluster them. As a result, the free cores can form a contiguous region. In [24], Pathania *et al*. introduced a concept of exponentially separable mapping (ESM), which defines a set of task mapping constraints on a many-core. They illustrated that an ESM-enforced many-core can be defragmented optimally in polynomial time. In [25], Wang *et al*. proposed a task migration-based adaptive tile (core) defragmentation algorithm that relocates the applications tile regions to consolidate running applications through online task migration. However, these defragmentation approaches do not consider thermal issues. In addition, most of them target 2D NoCs instead of 3D NoCs.

3 Problem Formulation

In this work, the 3D NoC system model as in [12] is adopted. Fig. 2 shows the architecture of a 3D mesh NoC with TSVs as the vertical links. As reported in [12, 22], TSV-based vertical links can often provide faster and more energy-efficient communication compared to horizontal links. The 3D NoC is modeled as a directed graph $G(C, L)$, where C is the set of cores and L is the set of links connecting the cores. The cores of the system model can run at various voltage/frequency (*V/F*) levels. The deterministic ZXY routing is used. The layer which lies closest to the heat sink is referred to as the bottom layer and the most distant one from the heat sink is referred to as the top layer. A centralized resource manager is designed to monitor the arrival of application, manage resources and perform application mapping in the NoC system. Table 1 summarizes the definitions of symbols used in this paper.

3.1 Application Model

Each application is modeled as a directed graph $A_i = (T_i, E_i)$, where T_i is the set of tasks of the application and E_i is the set of directed edges representing data communication amongst the tasks. Each task $t \in T_i$, has a weight equal to its execution time. Each edge $e \in E_i$ has a weight corresponding to the data traffic volume between the two communicating tasks.

A mapping function $M(t) = c$ binds tasks to the cores for each $t \in T_i$, and $c \in C$, such that task t is mapped to core c. Each edge $e \in E_i$ has a weight of transmission time, after the two communicating tasks are mapped. The transmission time between two tasks i and j depends on (1) the communication distance between the cores to which they are mapped and (2) their traffic volume. For each edge $e = (i, j)$, the transmission time can be modeled by equation 1.

$$Tr(e) = \alpha \cdot V(i, j) + \beta \cdot D(i, j) \quad (1)$$

where $V(i, j)$ is the traffic volume between the two tasks i and j, and $D(i, j)$ is the distance between the two cores to which the two tasks are mapped. $D(i, j)$ could only be computed after the two communicating tasks are mapped. α and β are regression coefficients of the linear regression model, which can be computed using the maximum likelihood method in [26]. The running time of each application i is the makespan of application A_i's task graph, denoted as RT_i.

3.2 Thermal Power Capacity Model

thermal power capacity model (TPC) of [27] is adopted. As in [27], the TPC of a core is defined as the maximum power the core can consume, given the power consumptions of other cores. It is used at runtime to estimate the upper bound power consumption of a core with low computing cost, given the power consumptions of its neighboring eight cores. The TPC of each core can be determined offline. In the rest of the paper, $P_M(c)$ and $P_M(x, y, z)$ are used to denote the power capacity of the core c at the location (x, y, z) interchangeably. The TPC of a core is bounded by the cooling capacity of the system, and the power consumptions

or temperatures of other cores, *i.e.*, thermal correlation. The TPC of a core c can be found as,

$$P_M(x,y,z) = \theta[P(x \pm l_1, y \pm l_2, z')] = \sum_q \alpha_q \cdot P(c) \quad (2)$$

where $P(x \pm l_1, y \pm l_2, z')$ is the power consumption of the core c located at $(x \pm l_1, y \pm l_2, z')$, which is thermally correlated with core c. The function $\theta(\cdot)$ can also be found by linear regression, using the lasso method [26]. In particular, for each core at (x, y, z) the coefficients of the following two types of cores are non-zero: (1) adjacent cores in the same layer, that is, cores whose coordinates are $(x \pm l_1, y \pm l_2, z')$ with l_1, l_2 = 0, 1 and $z' = z$ and (2) those with the same X and Y coordinates but in different layers, that is, cores whose coordinates are (x, y, z') with z' = 0, 1, ..., G_h and $z' \neq z$. These cores have the highest thermal correlations with the core (x, y, z). For core c, the coefficients of other cores are set to be 0.

3.3 Problem Description

thermal- and communication-aware mapping problem is defined as:

Given a set of n applications in the system, find a task-to-core mapping $M(A_i)$ for each application A_i in the 3D NoC system to minimize the overall running time of these applications, such that the system peak temperature is below a threshold.

Mathematically, the problem is formulated as:

$$\min \quad \sigma(S) = A_{lf} - A_{fa} \quad (3)$$

$$\text{subject to} \quad P(c) \leq P_M(c),\ \forall c \in C \quad (4)$$

where $\sigma(S)$ is the overall running time of the application set S, A_{lf} and A_{fa} are the finish time of the last application and the arrival time of the first application in the set, respectively, $P(c)$ is the power of a core, and $P_M(c)$ is the maximum power a core can consume which is computed from the TPC model.

4 The Proposed Thermal- and Communication-aware Mapping and Defragmentation Algorithm

4.1 Overview

The proposed mapping algorithm has three steps:

1) Find a 3D cuboid core region of a specific shape for every application.
2) Determine the exact location of each application's core region, only one task to one core mapping is allowed.
3) Perform defragmentation to keep the free core region contiguous.

4.1.1 Characterizing the shape of a core region

Two metrics, the minimal distance to heat sink (MD) and the number of occupied layers (NOL), are used to characterize the shape of a core region. Layers closer to the heat sink have a better cooling effect, which implies that cores in such layers can run with a higher *V/F* level and power consumption without violating thermal constraint. Therefore, the distance

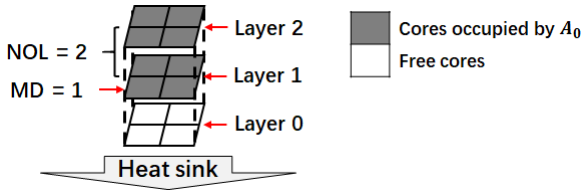

Fig. 3: An example of minimal distance to heat sink (MD) and the number of occupied layers (NOL) values of a core region.

to heat sink of a core region relates to the peak temperature and computation performance. The minimal distance to heat sink (MD) in definition 1 is used to characterize thermal feature of a core region. As an example, the MD value of the core region for the 8-task application in Fig. 3 is 1.

Definition 1. The minimal distance to heat sink (MD) is the index of the lowest layer of A_i's core region.

The number of occupied layers (NOL) of a core region, defined in definition 2, relates to the number of TSV links in a core region, corresponding to how many vertical links are used in a core region to accelerate communication. For an application whose number of tasks is fixed, a "taller" core region has more TSV links than a "shorter" one. Thus, a taller core region has lower network communication latency. That is, a larger "NOL" indicates lower communication latency. As an example, the NOL of the 8-task application in Fig. 3 is 2.

Definition 2. The number of occupied layers value NOL_i of application A_i's core region equals to its number of layers.

4.1.2 Estimation of an application's running time

The running time of an application depends on the communication latency and the *V/F* levels of the cores running its tasks. Cores near the heat sink can run at higher *V/F* levels to reduce task running time. MD metric can be used to reflect the computation performance of a core region. The communication latency depends on the average inter-task distance inside the region, as well as the number of vertical links inside a core region. Therefore, the running time of an application can be modeled by the NOL and MD metrics.

Consider the trade-off between model accuracy and algorithm runtime overhead, the running time of a core region can be estimated by a linear regression model of MD and NOL as in equation 5,

$$ERT_i = a_0 + a_1 \times |A_i| + a_2 \times V_i + a_3 \times NOL_i + a_4 \times MD_i \quad (5)$$

where $|A_i|$ is the number of tasks in application A_i, V_i is the average communication volume of each task in A_i, NOL_i is the number of occupied layers of the core region, and MD_i is the MD value of the core region. To compute the coefficients a_0, a_1, a_2, a_3, a_4, the maximum likelihood methods can be used [26].

A search algorithm is proposed where a core region of a specific shape is selected for each application, by tuning its MD and NOL values according to its communication

and computation demands. For a set of applications, a search tree is formed where each node in the search tree is characterized by the MD_i and NOL_i values of each application A_i.

The branch-and-bound algorithm is used to find the best tree node, which corresponds to the best combination of shapes of core region for this set of applications.

4.1.3 Search tree

Assume n applications are to be mapped, a tree node is defined as $N_m = \langle f(0), ..., f(n) \rangle$, where each $f(i)$ is composed by NOL_i and MD_i of application A_i's core region. At the root node, the MD and NOL values of the applications' core regions are not set yet.

The tree is grown by branching new nodes from the root down to the leaves level by level, corresponding to setting the MD_i and NOL_i values for each application's core region. Once the next level tree node is branched, the MD and NOL values of one of the applications' core region is set. Each non-leaf tree node N_j is associated with $\hat{\sigma}(S)_j^{MIN}$, indicating the minimal estimated running time of N_j. Each leaf node is associated with $\hat{\sigma}(S)$, that is, the time when the last application in set S finishes execution.

The basic operations for the search tree include branching and cutting.

4.1.3.1 *Branching of new tree nodes*: If a tree node is a leaf node, all the applications have found their core regions. The $\hat{\sigma}(S)$ of this tree node is compared with $\hat{\sigma}^*(S)$, a value keeps the minimum overall running time over all the tree nodes searched so far. If this tree node is not a leaf node, a new tree node should be created corresponding to finding a core region for a next application. The MD_i and NOL_i of the next application A_i are bounded by the following parameters.

$$NOL_i^{MIN} = \left\lceil \frac{|A_i|}{G_w \times G_l} \right\rceil \quad (6)$$

$$NOL_i^{MAX} = \min\{G_h, |A_i|\} \quad (7)$$

$$MD_i^{MIN} = \min\{c.z, \forall c \in \Gamma\} \quad (8)$$

$$MD_i^{MAX} = G_h - NOL_i^{MIN} \quad (9)$$

where $|A_i|$ is the number of tasks in A_i, Γ is the free core set, $c.z$ is the layer index of core c and G_w, G_l and G_h are the width, length and height of the 3D NoC system, respectively. NOL_i^{MIN} is the minimum number of occupied layers of a 3D cuboid core region for an application such that the application's tasks can be accommodated in the core region. NOL_i^{MAX} is the upper limit of the core region's number of occupied layers, which is bounded by the maximum number of vertical layers in the 3D NoC. MD_i^{MIN} and MD_i^{MAX} are the minimum and maximum MD values of a core region, respectively. Once NOL_i^{MIN} and NOL_i^{MAX} are determined, they can be computed to make sure the core region can fit inside the 3D NoC. The NOL and MD values of the next application's core region is bounded by $[NOL_i^{MIN}, NOL_i^{MAX}]$ and $[MD_i^{MIN}, MD_i^{MAX}]$, respectively. In total, for each non-leaf tree node, a maximum of $(NOL_i^{MAX} - NOL_i^{MIN}) \times (MD_i^{MAX} - MD_i^{MIN})$ tree nodes in the next level can be created.

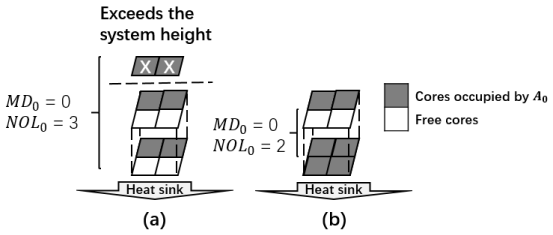

Fig. 4: (a) An example of infeasible core region. (b)An example of feasible core region.

4.1.3.2 *Cutting*: To reduce the search space and speed up the tree search process, some tree branches should be cut. For each of the newly created tree nodes, whether they should be discarded or not is checked according to the following two cut rules.

Rule 1) *Cut infeasible nodes.* An infeasible tree node refers to a setting of MDs and NOLs of the core regions that they do not fit into the 3D NoC, *e.g.*,

1) The number of cores of a region in a layer is larger than the total available cores in that layer. The number of available cores in each layer is maintained and updated at runtime to test the feasibility. Once some applications are mapped to run, the number of available cores in each layer is subtracted by the number of the cores occupied by the applications in that layer. The number of tasks in each layer of an application is set to be the number of tasks divided by NOL. For example, for a core region with an NOL of 2 and a 6-task application, the number of tasks in each layer in that core region is 3. If it exceeds the free core number limit in each layer, the corresponding tree node is infeasible.
2) For the vertical direction, it is feasible if $MD_i + NOL_i \leq G_h$, *i.e.*, the core region selected for the application fits inside the 3D NoC. For example, the core region in Fig. 4 (a) is infeasible because NOL_0 is larger than NoC height, causing two tasks exceeding the NoC. Meanwhile, the core region of A_0 in Fig. 4 (b) is feasible because all tasks of A_0 could be mapped inside the NoC.

The tree nodes not meeting these requirements are deemed as infeasible and discarded.

Rule 2) *Cut by node dominance.* Once a new tree node N_j is created, its $\hat{\sigma}(S)_j^{MIN}$ is compared with the global minimum overall running time $\hat{\sigma}^*(S)$. If its $\hat{\sigma}(S)_j^{MIN}$ is larger than $\hat{\sigma}^*(S)$, indicating the minimum overall application running time of N_j is longer than the best overall running time found so far among all the tree nodes, the new tree node is discarded.

Algorithm 1 shows how the search algorithm works. The tree nodes are stored in a working queue. Initially, a dummy root node is pushed to the queue. In each iteration, new tree nodes are created by assigning different MD and NOL values to a new application's core region. In total, $(NOL_i^{MAX} - NOL_i^{MIN}) \times (MD_i^{MAX} - MD_i^{MIN})$ new tree nodes are created. For each of these tree nodes, if they do not meet the above two cutting rules, they are pushed to the

Algorithm 1 Finding the Shape of Core Region for Each Application

Input: S: the set of unmapped applications;
Output: MD_i and NOL_i: MD and NOL values of each application A_i;
Var: WQ: A working queue, initialized to be empty;
NBN: The newly branched node;
BN: The best node during search;
$\hat{\sigma}^*(S)$: A value keeps the minimum overall running time over all the tree nodes searched so far;

```
while WQ is not empty do
    pop the top node N_q out of WQ;
    if N_q is not a leaf node then
        branch new tree nodes;
        for each newly branched nodes NBN do
            if NBN do not meet the cutting rules then
                push NBN in WQ;
        end
    else
        if σ̂(S)_q < σ̂*(S) then
            σ̂*(S) = σ̂(S)_q;
            BN = N_q;
end
```

queue. The length of the queue can be tuned to trade off the running speed and result optimality of the search algorithm. In the search process, applications with more traffic are possibly assigned with core regions with larger NOL values to speed up communication, while the computation intensive applications are possibly assigned with regions with smaller MD values to speed up computation.

4.2 Finding the Locations for the 3D Cuboid Core Regions

Algorithm 2 Finding the Exact Locations of Core Regions

Input: S: the set of unmapped applications;
Output: $M(A_i)$: the mapping core region for application A_i;
Function: Find the location of the core region for each application A_i;
Var: A_i: an application in S, where i = 1, 2, ..., m;

```
sort the applications in S according to their number of tasks in a descending order;
set CI_i to be 0;
for each A_i ∈ S do
    select the search starting core and search direction based on CI_i;
    search from the starting core along the corresponding direction, find an available core region;
    perform task-to-core mapping;
    CI_i = (CI_i + 1) % 4 ;
end
```

In this step, the exact location of each core region in the chip is found. The goals in this step are to 1) keep the applications scattered to reduce peak temperature, and 2) reduce fragmentation. To achieve these goals, free core regions are placed at one of the four corners of the chip in a round-robin manner. Algorithm 2 shows how the location finding step works.

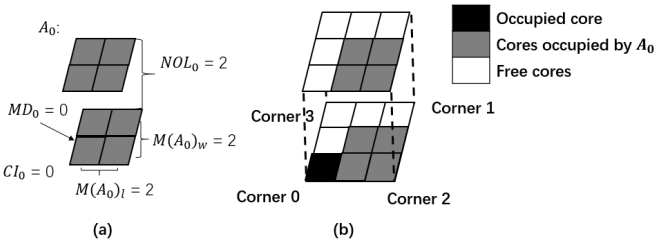

Fig. 5: (a) The mapping core region and other variables of A_0. (b) The four corners of example NoC and the mapping of A_0.

First, the applications are sorted according to their number of tasks in a descending order, *i.e.*, applications with more tasks are treated earlier. Next, the applications' core regions are placed in a round-robin manner to one of the four corners as in Fig. 5.b at each iteration. CI_i is used as the corner index for application A_i. Each corner is associated with a start point. $M(A_i)_w$ and $M(A_i)_l$, denoting the width and length of $M(A_i)$, are set to $\lceil\sqrt{\frac{|A_i|}{NOL_i}}\rceil$. Given a corner, start from its start point, scan along the X and Y directions to find a free 3D core region for A_i. The starting core and search direction of the 4 corners are determined as:

1) corner 0: Initialize its starting point as $(0, 0, MD_i)$, then search toward X+ direction until a free core region whose size is $M(A_i)_w \times M(A_i)_l \times MD_i$ is found. Otherwise, increase it's Y coordinate by 1 followed by scanning along X+ direction in search of such a free core region. This procedure continues until the desired core region is found or corner 3 is touched.
2) corner 1: Initialize its starting point as (G_w, G_l, MD_i), then search toward X- direction until a free core region whose size is $M(A_i)_w \times M(A_i)_l \times MD_i$ is found. Otherwise, decrease it's Y coordinate by 1 followed by scanning along X- direction in search of such a free core region. This procedure continues until the desired core region is found or corner 2 is touched.
3) corner 2: Initialize its starting point as $(G_w, 0, MD_i)$, then search toward Y+ direction until a free core region whose size is $M(A_i)_w \times M(A_i)_l \times MD_i$ is found. Otherwise, decrease it's X coordinate by 1 followed by scanning along Y+ direction in search of such a free core region. This procedure continues until the desired core region is found or corner 1 is touched.
4) corner 3: Initialize its starting point as $(G_w, 0, MD_i)$, then search toward Y+ direction until a free core region whose size is $M(A_i)_w \times M(A_i)_l \times MD_i$ is found. Otherwise, decrease it's X coordinate by 1 followed by scanning along Y+ direction in search of such a free core region. This procedure continues until the desired core region is found or corner 0 is touched.

An example mapping location searching procedure is

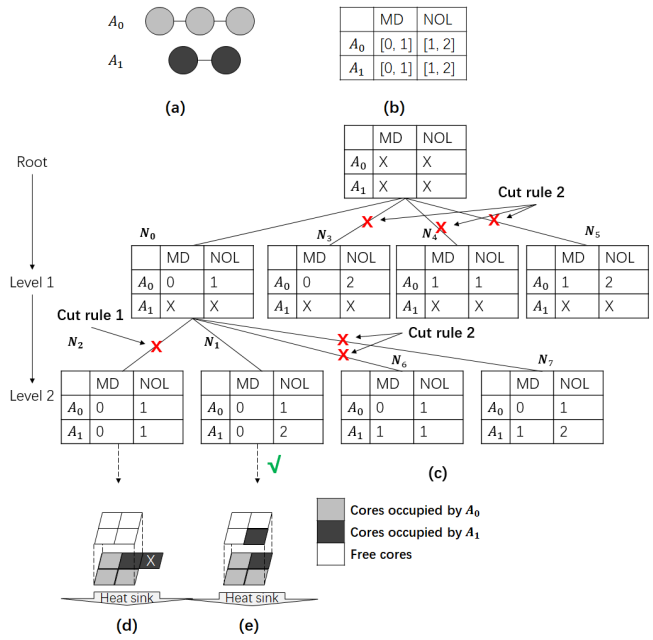

Fig. 6: (a) Two applications (A_0, A_1) to be mapped. (b) The ranges of MD and NOL values of the two applications. (c) Search tree in step 1. (d) An infeasible tree node. (e) System after mapping.

shown in Fig.5. A_0 is an 8-task application which NOL_0 is 2 and MD_0 is 0. The width and length of A_0's mapping core region shape, $M(A_0)_w$ and $M(A_0)_l$, are set to $\lceil\sqrt{\frac{|A_0|}{NOL_0}}\rceil$ and $\lceil\sqrt{\frac{|A_0|}{(NOL_0\times M(A_0)_w)}}\rceil$, which are 2. The height of A_0's mapping core region shape is equal to NOL_0, which is 2. Assumed that the mapping location searching procedure starts from corner 0, the CI of A_0, CI_0 is 0, and the searching started from corner 0. As Fig.5 (b) shows, corner 0 starts from Initialize its starting point as (0,0,0), then search toward along X+ direction, the searching direction is mentioned in section 4.2. The core $(0, 0, 0)$ is occupied, a mapping available core region of A_0 could not be found here. Then the searching procedure move one core ahead along X+ direction, and a mapping available core region of A_0 is found as in Fig.5.b.

In the last step, tasks of the application are mapped to the cores inside the free core region using existing mapping algorithms, for example, the one in [19]. This algorithm results in a contiguous free core region in the center of the chip. Therefore, fragmentation is alleviated. Besides, since the applications are mapped such that they are separated in the four corners, the peak temperature is also reduced.

4.3 An Example of the Mapping Algorithm

There are 2 applications A_0 and A_1 in a 2×2×2 3D NoC system to be mapped as shown in Fig. 6 (a). A_0 is a 3-task application and A_1 is a 2-task application. Fig. 6 (b) lists the MD and NOL ranges of each applications. Fig. 6 (c) is a snapshot of the search tree for finding the shapes of core regions for A_0 and A_1. Each tree node is represented by a 4-element vector $< NOL_0, MD_0, NOL_1, MD_1 >$. A dummy root node is first created. For A_0, a new tree node N_0 in level 1 is created from the root with $MD_0 = 0$ and $NOL_0 = 1$. Since A_0 has 3 tasks, it has 3 cores in layer 0. Next, a new node N_1 in level 2 is created for A_1 from N_0, with $MD_1 = 0$ and $NOL_1 = 2$. Since A_1 has 2 tasks, it has 1 core in both layers 0 and 1. Since the system has 4 cores in each layer, A_0 and A_1 fit into the system. Fig. 6 (d) shows an infeasible tree node N_2 in level 2. It requires 5 cores in layer 0 which exceeds the maximum number of available cores in layer 0. In Fig. 6(c), the ERT of N_1, $\hat{\sigma}(S)_1^{MIN}$, is kept as the global minimum overall running time $\hat{\sigma}^*(S)$. We assumed that the ERT of A_0 with tree node N_3-N_7 are larger than $^*(S)$. Under this assumption, the corresponding nodes in layer 1 are cut by cut rule 2. In level 2, the nodes are cut because their ERTs are larger than N_1's. Thus, N_1 is kept as the result of search.

Next, the locations of the applications' core regions are found with the MD_0, NOL_0, MD_1, NOL_1 values shown in N_1. Since A_0 has one more task than A_1, A_0 is mapped first. Since $MD_0 = 0$ and $NOL_0 = 1$, layer 0 needs 3 free cores for A_0. Starting from corner 1, scan along X+ direction and a cuboid of 3 available cores is found for A_0's core region. Now, for A_1, it needs 1 available core in both layers 0 and 1 for A_1. Start from corner 2, a cuboid of two available cores is found in each of the two layers for A_1. The locations of the two core regions are shown in Fig. 6 (e).

4.4 Thermal-aware Defragmentation

The arrival and departure of applications lead to fragmentation which leads to increased communication latency or waiting time. This causes degraded system throughput. To reduce fragmentation, a few defragmentation algorithms were proposed [6, 23, 24]. However, these algorithms ignore thermal constraint and might lead to overheating by clustering running cores into close proximity.

In this subsection, a thermal-aware defragmentation algorithm is proposed. Upon being triggered, this defragmentation algorithm tries to migrate the applications to the chip corners, keeping free cores clustered in the chip center as much as possible. As a result, a contiguous free core region can be formed in the center of the chip, which can accommodate subsequent incoming applications. In addition, the applications, which are scattered toward the four corners after defragmentation, tend to enjoy better cooling and lower temperature.

The defragmentation algorithm works as follows. A fragmentation metric is defined to measure the system fragmentation level. The defragmentation algorithm is triggered only when the fragmentation metric F is under a preset threshold F_{TH}. During the defragmentation process, a migration destination is found for each application, followed by the migration of the applications to the destination.

4.4.1 *Fragmentation metric*

The goal of the proposed defragmentation method is to move free cores to the center of the chip and form a contiguous region, by relocating the running applications to corners and edges of the chip. Therefore, the fragmentation metric is defined as the ratio of non-center free core region's size to

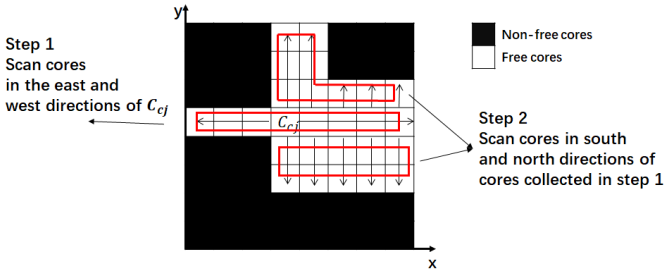

Fig. 7: Calculation of the number of free cores in center free core region in layer j.

the system's total number of free cores. A free core region's size is the number of free cores inside it. The fragmentation metric is defined as follows:

$$F = 1 - N_c/N_t \tag{10}$$

where N_c denotes the number of free cores in the center free core region and N_t denotes the total free core numbers in the system. N_c is calculated by algorithm 4. For each layer, the algorithm first finds the free cores in X direction, and then expand each found free core to find other free cores in Y direction. Fig. 7 illustrates the procedure in one layer. For each layer j, C_{cj} denotes the center core of that layer. In step 1, the cores in the east and west directions of C_{cj} are scanned to test if they are free. In step 2, for each found free core, cores in the south and north directions are scanned to test if they are free.

Algorithm 3 LINEFREECORECOUNT($C_{current}$, $Corrd_i$)

Input: $C_{current}$: the current core;
$Corrd_i$: the dimension to search, X or Y;
Output: $lineFreeCoreArray$: the number of free cores in the dimension to search;
Function: Calculate the number of free cores in the dimension to search;
Var: lineFreeCoreArray: a stack storing the free cores in the dimension to search;
initialize lineFreeCoreArray to be empty;
C_i = $C_{current}$
while C_i *is inside NoC and free* **do**
  move C_i one core ahead along dimension $Corrd_i$;
  push C_i into lineFreeCoreArray;
end
C_i = $C_{current}$
while C_i *is inside NoC and free* **do**
  move C_i one core backward along dimension $Corrd_i$;
  push C_i into lineFreeCoreArray;
end
return lineFreeCoreArray;

This fragmentation metric measures the percentage of free cores scattered outside the center free core region. If the system has a fragmentation metric larger than F_{TH}, a defragmentation process needs to be performed until it is under F_{TH}.

Algorithm 4 Calculating the Number of Free Cores in the Center Free Core Region

Input: S: the set of running applications;
C_{cj}: the center core of layer j;
Output: N_c: the number of free cores in the center free core region;
Function: Calculate the number of free cores in the center free core region;
Var: XfreeCoreArray: the stack storing the free cores in X dimension;
YfreeCoreArray: the stack storing the free cores in Y dimension;
freeCoreArray: the stack storing the free cores in the whole system;
initialize XfreeCoreArray to be empty;
for *each layer j in the system* **do**
  //Step 1
  XfreeCoreArray = LINEFREECORECOUNT(C_{cj}, X)
  //Step 2
  Initialize freeCoreArray and YFreeCoreArray to be empty;
  while *XfreeCoreArray is not empty* **do**
    pop a free core C_{fa} from XfreeCoreArray;//denote it as C_{fa}
    YFreeCoreArray = LINEFREECORECOUNT(C_{fa}, Y)
    push the elements of YFreeCoreArray into freeCoreArray
  end
  return the number of elements in freeCoreArray;
end

4.4.2 The defragmentation procedure

When the fragmentation metric F is larger than F_{TH}, there are a few fragments in the system. During the defragmentation process, in each iteration, only the application that is migrated are stopped execution. The other applications are unaffected, i.e., keep execution, until it is stopped and migrated in the next iteration. In this manner, the defragmentation has the minimal impact on the system execution, i.e., in each iteration only application is stopped and migrated. Therefore, the migration path has an impact on the destination selection of the migrating application. If the applications core region is blocked by another application during the migration, it has to stop, since the other application is still running. The intuition behind defragmentation is that, we maintained the center free core area by moving the applications running in the center to edges or corners to reduce fragmentation. Defragmentation would be triggered when the system has high fragmentation level. We used the fragmentation metric F to measure fragmentation level. Once it is over the threshold, the system start to calculate the migration destination and path, then moving the applications to edges and corners. The trigger condition is defined as follow:

$$F > F_{TH} \tag{11}$$

where F is the fragmentation metric of the running system and F_{TH} is the fragmentation threshold. Once the defragmentation is triggered, a destination is found for each

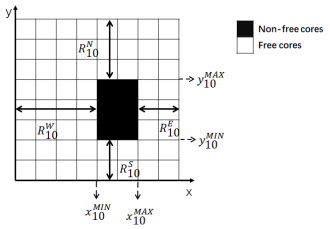

Fig. 8: The distances of A_1's core region toward the four edges of the chip in layer 0.

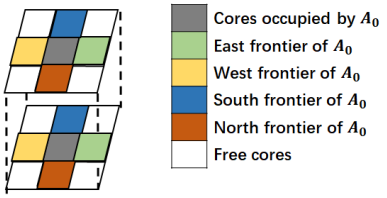

Fig. 9: The four frontiers of the application's core region.

application to migrate, followed by finding a migration path to each destination.

Applications are sorted by their number of tasks in a descending order. Applications with more tasks are migrated earlier. The defragmentation procedure works iteratively application by application until F is equal or smaller than F_{TH}, where each iteration includes two major steps for an application:

1) Determine the migration destination.
2) Find the migration path to the destination.

4.4.2.1 Determine the migration destination: We start by determining the order of doing defragmentation for every application. In the proposed algorithm, we serve the application with most tasks first, then serve other task number descending applications. In this way, the location of relatively big applications are determined earlier, the small one can be used to fill the gaps between them. After sorting the applications and put them in queue, the GM finds migration path for every running application. For each application, a nearest corner is selected as the destination. As shown in Fig. 8, each application has four parameters R^E_{ij}, R^S_{ij}, R^W_{ij} and R^N_{ij}, indicating the distances of A_i's core region toward the four edges of the chip in layer j, respectively. As in Fig. 8, x^{MAX}_j, x^{MIN}_j, y^{MAX}_j and y^{MIN}_j are the maximum x, minimum x, maximum y, minimum y coordinates of A_i's core region in layer j. R^E_{ij}, R^S_{ij}, R^W_{ij} and R^N_{ij} in layer j can be computed as equations 12, 13, 14, 15, respectively.

$$R^E_{ij} = G_w - x^{MAX}_j \tag{12}$$
$$R^W_{ij} = x^{MIN}_j \tag{13}$$
$$R^N_{ij} = G_l - y^{MAX}_j \tag{14}$$
$$R^S_{ij} = y^{MIN}_j \tag{15}$$

where G_w is the width (X dimension) of the NoC in each layer, and G_l is the length (Y dimension) of the NoC. The corner can be selected by the following rules.

1) If $R^N_{ij} > R^S_{ij}$ and $R^E_{ij} > R^W_{ij}$, select the south west corner as the destination.
2) If $R^N_{ij} < R^S_{ij}$ and $R^E_{ij} > R^W_{ij}$, select the north west corner as the destination.
3) If $R^N_{ij} > R^S_{ij}$ and $R^E_{ij} < R^W_{ij}$, select the south east corner as the destination.
4) If $R^N_{ij} < R^S_{ij}$ and $R^E_{ij} < R^W_{ij}$, select the north east corner as the destination.

4.4.2.2 Find the migration path: Once the destination corner for each application A_i is determined, the application can be moved towards it. For each application A_i in S, their destination corners are determined in order of corners 1, 2, 3, and 4 iteratively. The task closest to the destination corner is selected as reference task. A_i can migrate along one of the two virtual migration paths MP^{XY}_i and MP^{YX}_i, whereas in the former path, A_i moves along X direction first and then Y direction, while in the latter path moves along Y direction first and then X direction. For MP^{XY}_i, it checks the east (west) frontier (shown in Fig. 9) in the X+ (X-) direction. If the east (west) frontier is free, all the tasks of A_i can be virtually moved toward east (west) one hop further until no further virtual migration along the X direction is available. Next, it virtually moves toward the destination along the Y direction. it checks the north (south) frontier in the Y+ (Y-) direction. If the north (south) frontier is free, all the tasks of A_i can be virtually moved toward north (south) one hop further until it reaches the destination. For MP^{YX}_i, the tasks are virtually migrated in the same manner except that, it is first migrated along the Y direction and then X. Note that tasks are not really migrated along these two virtual paths at this time. The lengths of the two virtual paths MP^{XY}_i and MP^{YX}_i are denoted as $|MP^{XY}_i|$ and $|MP^{YX}_i|$, respectively. For one direction, the testing region $R_testing$ is first initialized to be the corresponding frontier of A_i as shown in Fig.9. Then it moves towards the direction one step at a time and pushes the core in the path into corresponding path until it hits non-free cores.

The lengths of the two virtual paths, $|MP^{XY}_i|$ and $|MP^{YX}_i|$ might differ, due to the fact that the application might be "blocked" by other applications on the way to the destination. For example, in Fig. 10, application A_1's destination is the south east corner. Along the MP^{YX}_i path, it can reach the destination (core 63). Its path length is 5. In contrast, along the MP^{XY}_i path, it cannot reach the destination due to occupied core 39 and the path length is 3. The defragmentation algorithm selects the longer path as the real migration path MP_i and all the tasks of the application are migrated along it. In this manner, the application can be migrated closer to a corner, leaving more free cores in the chip center.

Algorithm 6 shows how the defragmentation procedure works. For each A_i running in the system, the destination corner C_i is found first. Then the migration path MP_i is claculated by algorithm 5. Finally, the migration is performed along the selected migration path MP_i. All the tasks in its core region are moved simultaneously so that the shape of its core region is kept unchanged.

4.4.3 An example of the defragmentation process

Fig. 10 shows an example of how the defragmentation process works. Assume there are 2 applications, A_0 and A_1, and A_1 is to be migrated. R^N_{10}, R^E_{10}, R^S_{10}, R^W_{10} in layer 1 are

Algorithm 5 Finding Virtual Migration Paths for an Application

Input: $M(t_j)$: the core running the reference task t_j;
Output: MP_i^{XY}, MP_i^{YX}: the two virtual migration paths of A_i;
Function: Find virtual migrations path for A_i;
Var: $C_{current}$: the reference core to start the search process;
initialize MP_i^{XY} and MP_i^{YX} to be empty;
//find MP_i^{XY}
set $C_{current}$ to be $M(t_j)$;
while *the corresponding frontier of A_i is within the NoC system and cores inside this frontier are free* **do**
  move the core region of A_i and $C_{current}$ one hop ahead along X dimension, toward the direction selected by the rules in section 4.4.2.1;
  push $C_{current}$ into MP_i^{XY};
end
while *the corresponding frontier of A_i is within the NoC system and cores inside this frontier are free* **do**
  move the core region of A_i and $C_{current}$ one hop ahead along Y dimension, toward the direction selected by the rules in section 4.4.2.1;
  push $C_{current}$ into MP_i^{XY};
end
// find MP_i^{YX}
set $C_{current}$ to be $M(t_j)$;
while *the corresponding frontier of A_i is within the NoC system and cores inside this frontier are free* **do**
  move the core region of A_i and $C_{current}$ one hop ahead along Y dimension, toward the direction selected by the rules in section 4.4.2.1;
  push $C_{current}$ into MP_i^{YX};
end
while *the corresponding frontier of A_i is within the NoC system and cores inside this frontier are free* **do**
  move the core region of A_i and $C_{current}$ one hop ahead along X dimension, toward the direction selected by the rules in section 4.4.2.1;
  push $C_{current}$ into MP_i^{YX};
end
return MP_i^{XY}, MP_i^{YX}

Algorithm 6 Defragmentation

Input: S: the set of running applications;
Output: migration path of each application A_i in S;
Var: F: Fragmentation metric value of system;
  C_i: Destination corner of A_i;
  F_{TH}: Fragmentation metric threshold;
if $F > F_{TH}$ **then**
  sort the applications in S ny task number in a descending order;
  while S *is not empty* **do**
    pop A_i from S;
    find the destination corner C_i for A_i;
    find the migration path for A_i toward C_i, and migrate A_i along this path;
  end

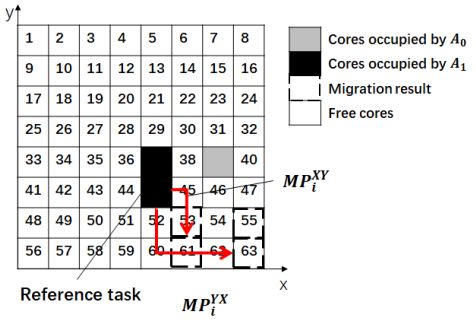

Fig. 10: Defragmentation of A_1. Only layer 1 is shown. The other layers resembles layer 1.

TABLE 2: Complexity Comparison

Mapping Algorithm Complexity	
	Worst Case Complexity
B2T[12]	$O(\|C\| \times \|A_i\|^2 \times \|E\|)$
FL[17]	$O(\|C\| \times \|A_i\|)$
Proposed	$O(\|WQ\| \times \|A_i\| \times \|E\| \times \|C\|)$
Defragmentation Algorithm Complexity	
	Worst Case Complexity
[6]	$O(\|C\|^2)$
[25]	$O(\max(\|WQ\| \times \|A_i\| \times \|C\|, N^3))$
Proposed	$O(\|A_i\| \times \|C\|)$

3, 2, 2 and 4, respectively. Thus, the south east corner (core 63) is selected as migration destination.

4.5 Complexity Analysis

Let $|C|$ denote the size of NoC, $|A_i|$ denote the number of tasks of application A_i, and $|WQ|$ denote the maximum length of the working queue WQ. The complexities of different mapping algorithms and defragmentation algorithms are listed in table 2.

5 Experimental Results

5.1 Experimental Setup

Experiments were performed on an event-driven C++ NoC simulator, with DSENT integrated as the power model. The simulator models the packet latency of the communications in a cycle accurate manner. In the experiments, two kinds of benchmarks are used: benchmarks whose task graphs are randomly generated and real benchmarks. The task graphs of the real applications are generated from the traces of SPLASH-2 [28] and PARSEC [29], which are collected by executing these applications in a 8×8 NoC-based cycle accurate many-core simulator [30]. The configuration of random benchmarks, real benchmarks, and the 3D NoC system are listed in Table 3. The temperature threshold is 60 °C. HotSpot [31] is used as the temperature simulator. The Hotspot parameters are shown in Table 3.

Our approach is compared with the following two runtime mapping approaches, the Bottom-2-Top (B2T) method [12] and the fuzzy logic (FL) method [17]. The B2T scheme first maps all the tasks to the bottom layer in a 3D NoC to

TABLE 3: Configurations of the Simulation

Network Parameters	
Flit size	128 bits
Latency Router	2 cycles, link 1 cycle
Buffer depth	4 flits
Routing algorithm	ZXY routing
Baseline topology	$8 \times 8 \times 4$
Random Benchmark Parameters	
Number of tasks	[15, 45]
Communication volume	[10, 200] (Kbits)
Degree of tasks	[1, 15]
Task number distribution	Bimodal, uniform
Configuration of the Many-core Simulator for Extracting Traces	
Core architecture	64 bit Alpha 21264
Baseline frequency	3GHz
Fetch/decode/commit size	4/4/4
ROB size	64
L1 D cache (private)	16KB, 2-way, 32B line 2 cycles, 2 ports, dual tags
L1 I cache (private)	32KB, 2-way 64B line, 2 cycles
L2 cache (shared) MESI protocol	64KB slice/core, 64B line 6 cycles, 2 ports
Main memory size	2GB
Task Graphs of Real Applications	
Barnes, Canneal, Raytrace, Dedup, Ferret, Freqmine Streamcluster, Fluidanimate, Swaptions, Blackscholes	
Hotspot Parameters	
Die size [mm]	0.5×0.5
Specific heat capacity [J/(m^3 × K)]	$1.75e^6$
Resistivity [(K · m)/W]	0.01
Layer 0 thickness [mm]	0.10
Layer 1 thickness [mm]	0.12
Layer 2 thickness [mm]	0.14
Layer 3 thickness [mm]	0.16

make the power consumption of cores in each vertical stack (cores with the same Z coordinate) identical as possible, and then moves low power tasks to the top layer to reduce the execution time [12]. FL defines three variables, *i.e.*, heat transfer, distance from source core, and distance from hot spot. It uses rules to set the priorities of them. "Distance from source core" represents the inter-task distance within the application. "Heat transfer" is the heat transfer capability of a specific core. "Distance from hot spot" is the distance from the hottest core. During the experiment, the order of the three variables are: (1) "distance from source core", (2) "heat transfer", and (3) "distance from hot spot". That is, FL maps communicating tasks to cores of shorter distance first. If such cores are not available, then the cores near the heat sink are used. If both of them are not available, cores far from the hot spot are used.

5.2 Testing Thermal Violation of TPC Model

We perform a set of experiments to test the probability of thermal violations (i.e., the peak temperature is over the temperature threshold). The experiments are performed as follows. N experiments are performed using TPC models in with each cores power consumption set randomly. We count the number of cases that the peak temperature is over the threshold. We define the probability of thermal violation P_{VT} as,

$$\epsilon = \frac{V_T}{N} \times 100\% \quad (16)$$

where V_T is the times of thermal violation, and N is the total experiment times. We run 100,000 experiments, where in each experiment, the power consumptions of the cores are randomly set except a particular core c_i. The maximum allowed power (TPC) of core c_i, $P_M(i)$, is computed by TPC model. The maximum power consumption of core c_i together with the power consumptions of other cores are feed into Hotspot as input power trace to calculate the temperature. After performing experiments for TPC models, we find that using TPC model leads to no thermal violation in our experiments.

5.3 Validating the Transmitting Time Model

The error of the transmitting time estimation model is defined as follow,

$$\epsilon = \frac{Tr(e) - \hat{Tr(e)}}{Tr(e)} \times 100\% \quad (17)$$

where $Tr(e)$ and $\hat{Tr(e)}$ are the transmission times obtained from NoC simulator and the transmission time model, respectively. We perform 10000 experiments to test the error of the transmission time model. The error of transmission time is 1.3% on average. Thus, the estimation is fairly accurate.

5.4 Validating the Application Running Time Model

The error of the application running time estimation model is defined as follow,

$$\epsilon = \frac{|RT - ERT|}{RT} \times 100\% \quad (18)$$

where RT and ERT are the running times obtained from the simulator and the estimation model in Equation 5 for each application, respectively. The error of this estimation model is 4.82% on average, for the applications used in the experiments. Thus, the estimation is fairly accurate.

5.5 Performance Comparison

5.5.1 Experiments with random benchmarks

5.5.1.1 *Experiments with different sizes of the 3D NoC*: To evaluate the proposed algorithm, the size of the 3D NoC is changed to evaluate the overall running time and communication latency of the three algorithms. Fig. 11 shows that our proposed algorithm outperforms the other two when the network size increases while running 100 applications. On average, the overall running times of B2T and FL are $1.39\times$ and $1.42\times$ over our proposed approach, respectively. Fig. 11 also shows that the communication latencies of B2T and FL are $1.55\times$ and $1.24\times$ over our proposed approach, respectively. The reason is that our algorithm can optimize the communication and computation performances for each application by selecting the appropriate MD and NOL values of its core region, according to its communication and computation demands. It can also reduce fragmentation and peak temperature by the core region location finding step. For every application,

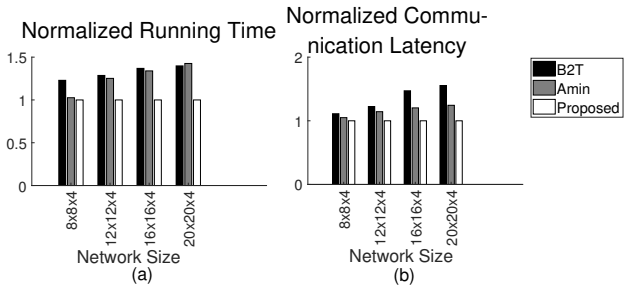

Fig. 11: (a) Normalized running time comparison with different sizes of the 3D NoC. (b) Normalized communication latency comparison with different sizes of the 3D NoC.

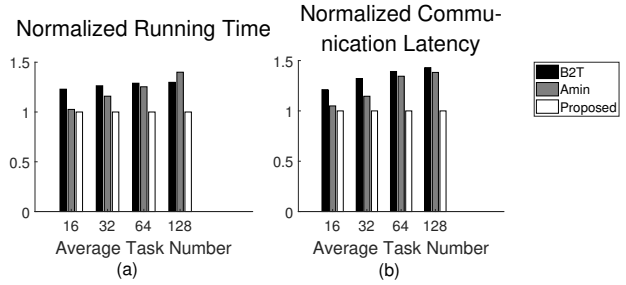

Fig. 12: (a) Normalized running time comparison with different task numbers of the applications. (b) Normalized communication latency comparison with different task numbers of the applications.

the B2T method makes the maximum use of the layer that is close to the heat sink and thus all the cores close to the heat sink are occupied and the incoming applications have to be mapped to other layers. Furthermore, B2T maps applications in close proximity, resulting in more heat accumulation and higher temperature.

5.5.1.2 *Experiments with different numbers of tasks*: In this set of experiments, the average number of tasks of the applications is changed. Fig. 12 shows that, when the average number of tasks is large, *e.g.* 128, the overall running times of B2T and FL are $1.29\times$ and $1.39\times$ over our proposed approach, respectively. Fig. 11 also shows that the communication latencies of B2T and FL are $1.42\times$ and $1.38\times$ over our proposed approach, respectively.

5.5.1.3 *Experiments with different communication volumes*: In this set of experiments, the average communication volume among the tasks of the applications is changed. Fig. 13 shows that, when the communication volume is high, *e.g.* 200 Kb, the overall running time of B2T and FL are $1.38\times$ and $1.25\times$ over our proposed approach, respectively. Fig. 11 also shows that the communication latencies of B2T and FL are $1.47\times$ and $1.13\times$ over our proposed approach, respectively.

5.5.2 Experiments with real benchmarks

To evaluate the proposed mapping algorithm, the performances on real benchmarks with 4 different network sizes are compared by running 10 applications. For each experiment, a mix of 100 applications were used as input where

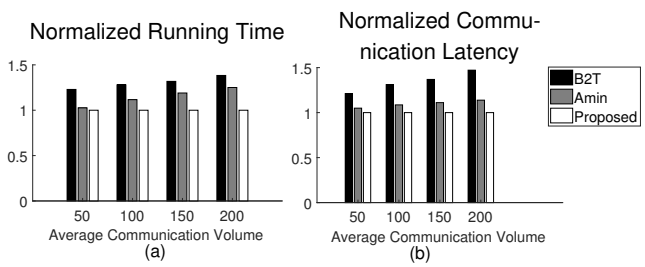

Fig. 13: (a) Normalized running time comparison with different communication volumes of the applications. (b) Normalized communication latency comparison with different communication volumes of the applications.

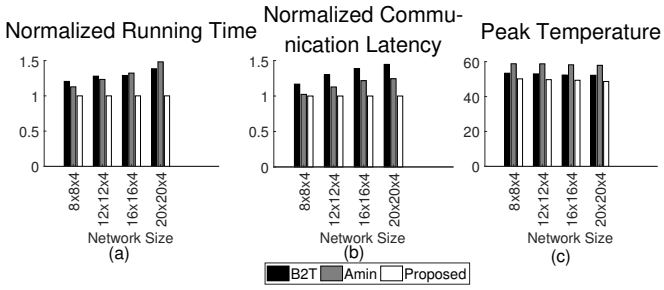

Fig. 14: (a) Normalized running time comparison with different sizes of the 3D NoC. (b) Normalized communication latency comparison with different sizes of the 3D NoC. (c) Peak temperature comparison with different sizes of the 3D NoC.

the applications are repeatedly picked from the 10 real benchmarks. Fig. 14 (a) shows that, when the network size is large, *e.g.* $20 \times 20 \times 4$, the overall running times of B2T and FL are $1.38\times$ and $1.48\times$ over our proposed approach, respectively. Fig. 14 also shows that the communication latencies of B2T and FL are $1.44\times$ and $1.24\times$ over our proposed approach, respectively. Fig. 14 also compares the peak temperatures of different algorithms, showing that our algorithm reduces the peak temperatures of B2T and FL by 3°C and 7°C. All of the three mapping algorithms are below the 60°C thermal threshold. The normalized communication latency and running time of each real benchmark application are compared, as shown in Fig. 15. Our proposed algorithm reduces the communication latency by 32% and

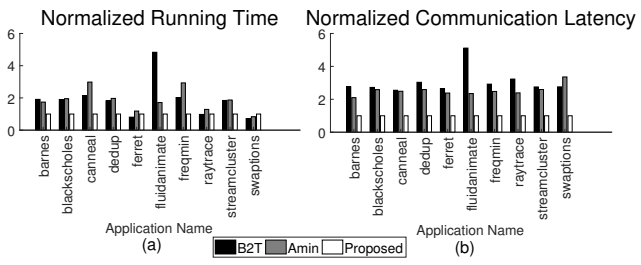

Fig. 15: (a) Normalized running time comparison with each real benchmark. (b)Normalized communication latency comparison with each real benchmark.

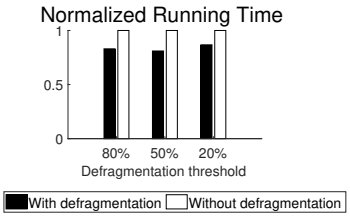

Fig. 16: Normalized running time comparison with and without defragmentation by varying the fragmentation threshold.

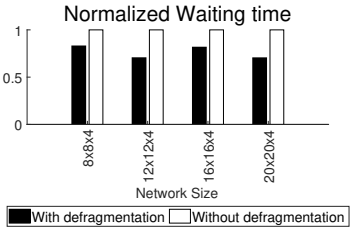

Fig. 17: Normailzed waiting time comparison with and without defragmentation by varying the size of the 3D NoC.

15% over B2T and FL on average, respectively.

5.5.3 Evaluation of fragmentation metric threshold

In section 4.4, the proposed defragmentation algorithm is triggered when F is larger than F_{TH}. In other words, the threshold controls the defragmentation frequency. The impact of F_{TH} on the system performance is evaluated in Fig. 16. Fig. 16 shows that, the proposed defragmentation process reduces the running time by as much as 19% when the F_{TH} is set to be 50%. Therefore, the threshold is set to be 50%.

5.5.4 Evaluation of the defragmentation algorithm

5.5.4.1 *Experiments with evaluating defragmentation with random benchmarks*: An experiment measuring the average waiting time by running 100 applications (each with 16 tasks on average) with different sizes of the NoC was conducted and the results are presented in Fig. 17. Waiting time of an application is measured as the time elapsed between the time it arrives at the system and the time when it can run on the cores. The result shows that the average waiting time of each application can be saved up to 29% with the proposed defragmentation process.

To evaluate the performance of the defragmentation process, we compare the performances of the previously proposed Defrag [6] algorithm, our proposed defragmentation algorithm, and the original system without performing any defragmentation algorithm. Using Defrag [6], the shapes of the applications' core regions might be irregular or even scattered far away, leading to higher communication cost. Also Defrag gathers running cores together which causes higher temperature. Fig. 18 (a) shows the normalized running time comparison with different defragmentation methods by varying the size of the 3D NoC. With the proposed defragmentation process, the running time is reduced by as much as 21% compared to the original mapping algorithm without defragmentation, and the running time of Defrag is increased by as much as 14% over the original mapping algorithm due to higher communication cost. Fig. 18 (b) shows the normalized running time comparison with different defragmentation methods by varying the average task number of applications. With the proposed defragmentation process, the running time is reduced by as much as 25% compared to the original mapping algorithm without defragmentation, and the running time of Defrag is increased by 7% over the original mapping algorithm.

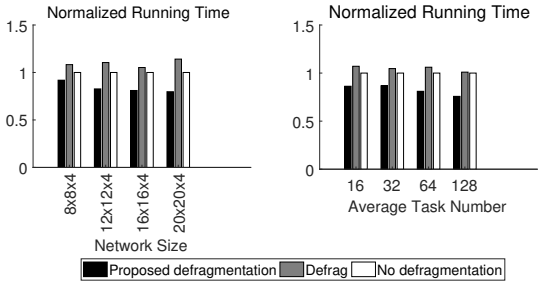

Fig. 18: (a) Normalized running time comparison with different defragmentation methods by varying the size of 3D NoC. (b) Normalized running time comparison with different defragmentation methods by varying the average task number of applications.

5.5.4.2 *Experiments evaluating defragmentation with real benchmarks*: This experiment evaluates the performances of Defrag [6], the proposal defragmentation algorithm, and the original mapping without performing any defragmentation algorithm. Fig. 19 (a) shows that, with the proposed defragmentation, the running time can be reduced by as much as 32% compared to the original mapping algorithm without defragmentation, and the running time of Defrag is increased by as much as 50% over the original mapping algorithm. Fig. 19 (b) shows that, the peak temperature with the proposed defragmentation is around 2°C higher than that without defragmentation due to the higher utilization of system cores and shorter running time. The peak temperature of Defrag is 2.5°C higher than the case without defragmentation as Fig. 19 shows. However, no thermal threshold violation is observed in the experiments. The peak temperature is under the 60°C threshold.

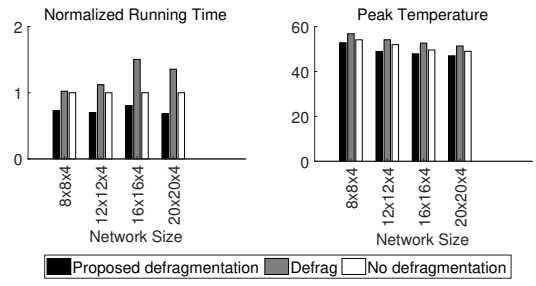

Fig. 19: (a) Normalized running time comparison with real benchmarks. (b) Peak temperature comparison with real benchmarks.

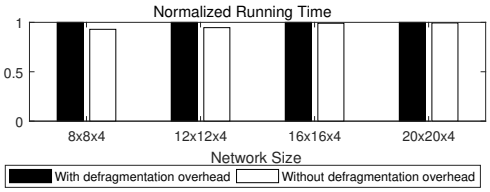

Fig. 20: Normalized running time comparison with and without defragmentation overhead

5.5.5 Runtime overhead of the proposed algorithm

The runtime overhead includes the overhead of the mapping algorithm and defragmentation algorithm.

The average runtime overheads of B2T, FL and our mapping algorithm are in the order of 3.5M, 5M and 4M cycles for one real benchmark on average, which are averaged by running each of the algorithms for fifty times with different real benchmark applications. Experimental results have shown that the execution time of each application is about 50M-100M cycles which is much longer than 4M cycles. We perform experiments to compare the normalized running time comparison with and without defragmentation overhead. The results in Fig.20 showed that, defragmentation overhead has little influence on system performance and is acceptable.

The defragmentation overhead includes the algorithm running time and the time spent in task migration. Experimental results show that the average running time is about 0.9M cycles for one defragmentation process and the average migration overhead is around 0.06M cycles. In the experiments, the fragmentation process is triggered once per 1000M cycles on average. Comparing with the average application execution time, the overhead of the proposed defragmentation algorithm is acceptable.

6 Conclusion

In this paper, we proposed a runtime communication- and thermal-aware mapping and defragmentation algorithm to optimize performance under thermal constraint in 3D NoCs. This algorithm has three steps. First, the shape of the core regions are selected by setting the MD and NOL parameters for each of the application, according to its communication and computation demands. Second, the locations of the core regions in the chip are found, followed by a task-to-core mapping process. In addition, a thermal-aware defragmentation algorithm is further proposed to reduce system fragmentation and waiting time for incoming application under thermal constraint. Experimental results show that our proposed approach can reduce up to 48% overall running time compared to existing mapping algorithms. The defragmentation procedure can saved application running time by up to 32% overall running time without defragmentation. Thus the proposed algorithm is suitable for future many core systems performance optimization.

Acknowledgments

This research program is supported by the Natural Science Foundation of China No. 61376024 and 61306024, Natural Science Foundation of Guangdong Province 2015A030313743 and 2018A030313166, Special Program for Applied Research on Super Computation of the NSFC-Guangdong Joint Fund (the second phase), and the Science and Technology Research Grant of Guangdong Province No. 2016A010101011 and 2017A050501003, Pearl River S&T Nova Program of Guangzhou No. 201806010038, and Tip-top Scientific and Technical Innovative Youth Talents of Guangdong special support program (No. 2014TQ01X590).

References

[1] M. Motoyoshi, "Through silicon via (TSV)," *Proc. the IEEE*, vol. 97, no. 1, pp. 43–48, 2009.

[2] G. Katti, M. Stucchi, K. De Meyer, and W. Dehaene, "Electrical modeling and characterization of through silicon via for three-dimensional ICs," *IEEE Trans. Electron Devices*, vol. 57, no. 1, pp. 256–262, 2010.

[3] V. F. Pavlidis and E. G. Friedman, "3D topologies for networks-on-chip," *IEEE Trans. Very Large Scale Integration Systems*, vol. 15, no. 10, pp. 1081–1090, 2007.

[4] D. Park, S. Eachempati, R. Das, A. K. Mishra, Y. Xie, N. Vijaykrishnan, and C. R. Das, "Mira: a multi-layered on-chip interconnect router architecture," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3, 2008, pp. 251–261.

[5] C.-H. Chao, K.-Y. Jheng, H.-Y. Wang, J.-C. Wu, and A.-Y. Wu, "Traffic-and thermal-aware run-time thermal management scheme for 3D NoC systems," in *Proc. ACM/IEEE Int'l Symp. Networks-on-Chip*, 2010, pp. 223–230.

[6] J. Ng, X. Wang, A. K. Singh, and T. Mak, "DeFrag: defragmentation for efficient runtime resource allocation in NoC-based many-core systems," in *Proc. Int'l Conf. Parallel, Distributed and Network-Based Processing*, 2015, pp. 345–352.

[7] B. Li, X. Wang, A. K. Singh, and T. Mak, "On runtime communication-and thermal-aware application mapping in 3D NoC," in *Proc. IEEE/ACM Int'l Symp. Networks-on-Chip*, 2017, pp. 16:1–16:8.

[8] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris, "Distributed run-time resource management for malleable applications on many-core platforms," in *Proc. Design Automation Conf.*, 2013, pp. 168:1–6.

[9] H. Ziaeeziabari and A. Patooghy, "3D-AMAP: a latency-aware task mapping onto 3D mesh-based NoCs with partially-filled TSVs," in *Proc. Int'l Conf. Parallel, Distributed and Network-based Processing*, 2017, pp. 593–597.

[10] E. L. de Souza Carvalho, N. L. V. Calazans, and F. G. Moraes, "Dynamic task mapping for MPSoCs," *IEEE Design & Test of Computers*, vol. 27, no. 5, pp. 26–35, 2010.

[11] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proc. Design Automation Conf.*, 2013, pp. 1–6.

[12] Y. Cui, W. Zhang, V. Chaturvedi, W. Liu, and B. He, "Thermal-aware task scheduling for 3D network-on-

chip: a bottom to top scheme," *J. Circuits, Systems and Computers*, vol. 25, no. 1, pp. 224–227, 2016.
[13] C. Sun, L. Shang, and R. P. Dick, "Three-dimensional multiprocessor system-on-chip thermal optimization," in *Proc. IEEE/ACM Int'l Conf. Hardware/software Codesign and System Synthesis*, 2007, pp. 117–122.
[14] C. Zhu, Z. Gu, L. Shang, R. P. Dick, and R. Joseph, "Three-dimensional chip-multiprocessor run-time thermal management," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 8, pp. 1479–1492, 2008.
[15] D. Zhu, L. Chen, T. M. Pinkston, and M. Pedram, "Tapp: temperature-aware application mapping for noc-based many-core processors," in *Proc. Design, Automation & Test in Europe Conf. & Exhibition*, 2015, pp. 1241–1244.
[16] P. K. Hamedani, S. Hessabi, H. Sarbazi-Azad, and N. E. Jerger, "Exploration of temperature constraints for thermal aware mapping of 3D networks on chip," in *Proc. Euromicro Int'l Conf. Parallel, Distributed and Network-Based Processing (PDP)*, 2012, pp. 499–506.
[17] A. Mosayyebzadeh, M. M. Amiraski, and S. Hessabi, "Thermal and power aware task mapping on 3D network on chip," *Computers & Electrical Engineering*, vol. 51, pp. 157–167, 2016.
[18] J. Li, M. Qiu, J.-W. Niu, L. T. Yang, Y. Zhu, and Z. Ming, "Thermal-aware task scheduling in 3D chip multiprocessor with real-time constrained workloads," *ACM Trans. Embedded Computing Systems*, vol. 12, no. 2, p. 24:122, 2013.
[19] X.-H. Wang, P. Liu, M. Yang, M. Palesi, Y.-T. Jiang, and M. C. Huang, "Energy efficient run-time incremental mapping for 3D networks-on-chip," *J. Computer Science and Technology*, vol. 28, no. 1, pp. 54–71, 2013.
[20] Z. Zhu, V. Chaturvedi, A. K. Singh, W. Zhang, and Y. Cui, "Two-stage thermal-aware scheduling of task graphs on 3D multi-cores exploiting application and architecture characteristics," in *Proc. Asia and South Pacific Design Automation Conf.*, 2017, pp. 324–329.
[21] C. Addo-Quaye, "Thermal-aware mapping and placement for 3D NoC designs," in *Proc. IEEE Int'l SoC Conf.*, 2005, pp. 25–28.
[22] M. Cox, A. K. Singh, A. Kumar, and H. Corporaal, "Thermal-aware mapping of streaming applications on 3D multi-processor systems," in *Proc. Symp. Embedded Systems for Real-time Multimedia*, 2013, pp. 11–20.
[23] F. G. Moraes, G. A. Madalozzo, G. M. Castilhos, and E. A. Carara, "Proposal and evaluation of a task migration protocol for noc-based mpsocs," in *Int'l Symp. Circuits and Systems (ISCAS)*, 2012, pp. 644–647.
[24] A. Pathania, V. Venkataramani, M. Shafique, T. Mitra, and J. Henkel, "Defragmentation of tasks in many-core architecture," *ACM Trans. Architecture and Code Optimization (TACO)*, vol. 14, no. 1, pp. 2:1–21, 2017.
[25] X. Wang, T. Fei, B. Zhang, and T. Mak, "On runtime adaptive tile defragmentation for resource management in many-core systems," *Microprocessors and Microsystems*, vol. 46, pp. 161–174, 2016.
[26] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*. Springer series in statistics New York, 2001, vol. 1.
[27] X. Wang, A. K. Singh, B. Li, Y. Yang, T. Mak, and H. Li, "Bubble budgeting: Throughput optimization for dynamic workloads by exploiting dark cores in many core systems," in *Proc. IEEE/ACM Int'l Symp. Networks-on-Chip*, 2016, pp. 1–8.
[28] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *Proc. Int'l Symp. Computer Architecture*, 1995, pp. 24–36.
[29] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: Characterization and architectural implications," in *Proc. Int'l Conf. Parallel architectures and compilation techniques*, 2008, pp. 72–81.
[30] X. Wang, M. Yang, Y. Jiang, P. Liu, M. Daneshtalab, M. Palesi, and T. Mak, "On self-tuning networks-on-chip for dynamic network flow dominance adaptation," *ACM Trans. Embedded Computing Systems*, vol. 13, no. 2, pp. 1–8, 2014.
[31] W. Huang, S. Ghosh, S. Velusamy, K. Sankaranarayanan, K. Skadron, and M. R. Stan, "HotSpot: a compact thermal modeling methodology for early-stage VLSI design," *IEEE Trans. Very Large Scale Integration Systems*, vol. 14, no. 5, pp. 501–513, 2006.

Bing Li received the bachelor degree in Software Engineering from South China University of Technology, Guangzhou, China. She is pursuing her master degree in the School of Software Engineering, South China University of Technology . Her research interest is task mapping for NoC-based systems.

Xiaohang Wang received the B.Eng. and Ph.D degree in communication and electronic engineering from Zhejiang University, in 2006 and 2011. He is currently an associate professor at South China University of Technology. He was the receipt of PDP 2015 and VLSI-SoC 2014 Best Paper Awards. His research interests include many-core architecture, power efficient architectures, optimal control, and NoC-based systems.

Amit Kumar Singh received the B. Tech. degree in Electronics Engineering from Indian Institute of Technology (Indian School of Mines), Dhanbad, India, in 2006, and the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University (NTU), Singapore, in 2013. He was with HCL Technologies, India for year and half before starting his PhD at NTU, Singapore, in 2008. He worked as a post-doctoral researcher at National University of Singapore (NUS) from 2012 to 2014, at University of York, UK from 2014 to 2016 and at University of Southampton, UK from 2016 to 2017. Currently, he is a Lecturer at University of Essex, UK. His current research interests include system level design-time and run-time optimizations of 2D and 3D multi-core systems with focus on performance, energy, temperature, and reliability. He has published over 60 papers in the above areas in leading international journals/conferences. Dr. Singh was the receipt of ICCES 2017 Best Paper Award, ISORC 2016 Best Paper Award, PDP 2015 Best Paper Award, HiPEAC Paper Award, and GLSVLSI 2014 Best Paper Candidate. He has served on the TPC of IEEE/ACM conferences like ISED, MES, NoCArc, ESTIMedia, ICESS, and DATE.

Terrence Mak is an Associate Professor at Electronics and Computer Science, University of Southampton. Supported by the Royal Society, he was a Visiting Scientist at Massachusetts Institute of Technology during 2010, and also, affiliated with the Chinese Academy of Sciences as a Visiting Professor since 2013. Previously, He worked with Turing Award holder Prof. Ivan Sutherland, at Sun Lab in California and has awarded Croucher Foundation scholar. His newly proposed approaches, using runtime optimisation and adaptation, strengthened network reliability, reduced power dissipations and significantly improved overall on-chip communication performances. Throughout a spectrum of novel methodologies, including regulating traffic dynamics using network-on-chips, enabling unprecedented MTBF and to provide better on-chip efficiencies, and proposed a novel garbage collections methods, defragmentation, together led to three prestigious best paper awards at DATE 2011, IEEE/ACM VLSI-SoC 2014 and IEEE PDP 2015, respectively. More recently, his newly published journal based on 3D adaptation and deadlock-free routing has awarded the prestigious 2015 IET Computers & Digital Techniques Premium Award. He has published more than 100 papers in both conferences and journals and jointly published 4 books.