Bubble Budgeting: Throughput Optimization for Dynamic Workloads by Exploiting Dark Cores in Many Core Systems

Xiaohang Wang, *Member, IEEE,* Amit Kumar Singh, *Member, IEEE,* Bing Li, Yang Yang, Hong Li and Terrence Mak, *Member, IEEE,*

Abstract—All the cores of a many-core chip cannot be active at the same time, due to reasons like low CPU utilization in server systems and limited power budget in dark silicon era. These free cores (referred to as bubbles) can be placed near active cores for heat dissipation so that the active cores can run at a higher frequency level, boosting the performance of applications that run on active cores. Budgeting inactive cores (bubbles) to applications to boost performance has the following three challenges. First, the number of bubbles varies due to open workloads. Second, communication distance increases when a bubble is inserted between two communicating tasks (a task is a thread or process of a parallel application), leading to performance degradation. Third, budgeting too many bubbles as coolers to running applications leads to insufficient cores for future applications. In order to address these challenges, in this paper, a bubble budgeting scheme is proposed to budget free cores to each application so as to optimize the throughput of the whole system. Throughput of the system depends on the execution time of each application and the waiting time incurred for newly arrived applications. Essentially, the proposed algorithm determines the number and locations of bubbles to optimize the performance and waiting time of each application, followed by tasks of each application being mapped to a core region. A Rollout algorithm is used to budget power to the cores as the last step. Experiments show that our approach achieves 50% higher throughput when compared to state-of-the-art thermal-aware runtime task mapping approaches. The runtime overhead of the proposed algorithm is in the order of 1M cycles, making it an efficient runtime task management method for large-scale many-core systems.

Index Terms—Online task management, power budget, dark silicon, many-core, dynamic resource allocation, temperature constraint, dark cores, throughput optimization, frequency scaling.

1 Introduction

Many-core chips are widely used in servers, datacenters, clusters to provide high throughput computation services. In such systems, applications or user requests dynamically arrive and leave the system without prior knowledge of future arrivals, which are referred as open systems [12]. One phenomenon observed in such high-performance many-core system is that, there are plenty of free cores which are either not utilized or even shut down from time to time. We have referred these free and powered-off cores as *dark cores* or *bubbles*. Free cores exist due to two reasons. First, in datacenters, the CPU usage is lower than 100% at most of the time, the average CPU utilization is as low as 50%, as shown in Figure 1 [13]. Therefore, some cores are not running useful applications at certain time period. Second, the high density integration of cores within chips leads to a possible dark silicon issue [17], where a large portion of the cores have to be turned off to meet the thermal and power constraints.

Several efforts have been made to exploit the bubbles (dark cores) to boost performance of active cores and applications, by determining the number, position, and voltage/frequency levels of the active cores [18], [25]. An active core can run at a higher level of frequency if bubbles are located near it for heat dissipation. This helps to achieve higher performance while meeting the temperature constraint. However, in a server system with open workloads (workloads are defined as application programs or user requests submitted to the many-core systems), the following challenges need to be addressed so as to optimize the overall system performance.

First, for a system handling open workloads, since the number of available/free cores changes with the arrival and departure of applications, the position of bubbles and voltage/frequency of active cores need to be adjusted at runtime under the temperature constraint in response to arrival and departure of applications. Most of the approaches (e.g., [18], [25]) consider static workloads only, i.e., a fixed set of applications known in advance and fixed number of bub-

- *X. Wang, B. Li, and H. Li are with the School of Software Engineering, South China University of Technology, China*
  Email: xiaohangwang@scut.edu.cn, l.b07@mail.scut.edu.cn, hongli@scut.edu.cn
- *A.K. Singh is with the School of Electronics and Computer Science, University of Southampton, UK*
  Email: a.k.singh@soton.ac.uk
- *Y. Yang is with the School of Data and Computer Science, Sun Yat-sen University, China*
  Email: yangy266@mail.sysu.edu.cn
- *T. Mak is with the School of Electronics and Computer Science, University of Southampton, UK, Shenzhen Institute of Advanced Technology and Guangzhou Institute of Advanced Technology, CAS, China*
  Email: tmak@ecs.soton.ac.uk

This research program is supported by the Natural Science Foundation of China No. 61376024 and 61306024, Natural Science Foundation of Guangdong Province 2015A030313743, 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.

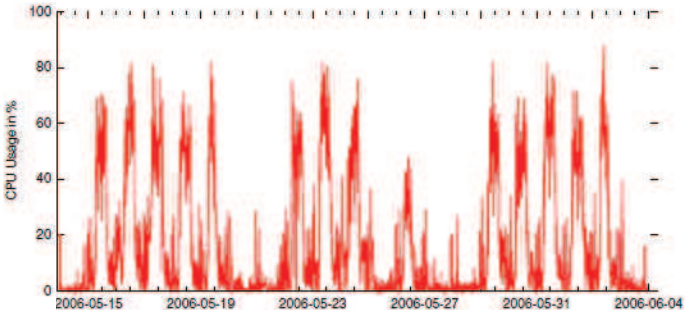


Fig. 1. CPU usage over several weeks [13].

bles, which does not reflect the dynamic feature of several systems, e.g., a server.

Next, communication overhead between the active cores executing communicating tasks is largely affected by placing bubbles near them. Communication distance between two tasks increases if the corresponding two cores have bubbles (dark cores) inserted between them for heat dissipation. Therefore, although active cores can possibly run at a higher frequency level if bubbles are placed near them, the applications might suffer from increased communication overhead, resulting in poor performance. Existing approaches (e.g., [18], [25]) ignore such communication overhead.

Furthermore, if most of the bubbles are placed near active cores and used for heat dissipation, a newly arrived application might need to wait for a longer time due to insufficient free cores. Therefore, the decision of whether a free core should be shut down for heat dissipation as a bubble, or to be turned on to run tasks affects both the execution time of current application and the possible waiting time for future applications. Existing approaches ignore waiting time incurred for each newly arrived application, which also affects the overall system throughput.

Contribution: This paper addresses the aforementioned challenges by proposing a lightweight dynamic resource management approach that handles open workloads, where applications containing dependent tasks arrive at different moments of time. The tasks are assumed as fixed, i.e. not malleable. This work tries to determine the number and location of both free and active cores, so as to optimize performance, communication cost and waiting time. A preliminary version of this work has been published in [40], which has the following main contributions:

1) Performance and waiting time models targeting open workloads, where applications arrive and depart in the system at different times. Therefore, the number of free cores vary in the system. These models can be updated online.
2) An online algorithm to select the number and locations of free cores for each application. Instead of optimizing each individual application's performance, this algorithm tries to optimize the system throughput in terms of number of executed applications within a given time, which depends on the waiting time for each newly arrived application and the execution time of each application. Both computation and communication performances are optimized when determining the number and location of bubbles and active cores.

Our previous work [40] has been significantly extended by making the following new contributions:

1) A detailed thermal model is provided.
2) A power budgeting algorithm is provided which serves as the last step of the whole online algorithm. This algorithm is based on the Rollout algorithm [5]. In the Rollout algorithm, when it is to make a decision at each step, a heuristic is used to estimate the cost of all possible options at the next step, referred as look ahead. The decision is made based on the estimates of the next step. It tunes the voltage/frequency of the cores after their locations are determined.
3) The experimental results are substantially updated while applying the new algorithm. Specially, results for a variety of network sizes and communication volumes are presented.

2 Related Work

Allocating system resources to the tasks of multiple applications on on-chip many-core system has been an emerging research direction [29], [35], [36]. Several resource allocation approaches have been proposed while following different policies. Most of these approaches map communicating tasks of each application close to each other such that communication overhead and power are reduced [2], [7], [8], [9], [11], [21], [24], [33], [37]. Some of these approaches also reduce computation power of the cores by employing voltage/frequency scaling [9]. However, these approaches do not consider a power budget for the whole chip, which is desired in the dark silicon era.

There has been some efforts to perform the mapping by taking the power budget into account [19], [23], [30], [38]. Some of these efforts just try to respect the power budget, whereas others try for the thermal design power budget. However, using a single and constant value as a power constraint for each core or for the entire chip in the mapping process may result in either thermal violation (*e.g.*, the peak temperature is over a safe threshold), or tremendous performance losses for many-core systems [25], [32]. Therefore, temperature of the cores needs to considered in order to avoid the thermal violations.

Thermal-aware resource allocation approaches have been explored to reduce peak temperature and temperature gradient while directly considering the temperature of cores [10], [27]. However, these approaches do not impose any thermal constraint in the allocation process. Some approaches considering thermal constraint while optimizing for the performance have been reported [15], [28]. However, in [15], heat conductance amongst the neighboring cores is ignored to simplify the problem and [28] considers only one application. Further, these thermal-aware resource allocation approaches do not consider dark silicon problem.

To address dark silicon problem while considering multiple applications, recently, some resource allocation approaches have been introduced [23], [25]. The approach in [25] identifies the number, location and voltage/frequency levels of active cores for each application to optimize the overall system performance. It also leverages the positioning of dark cores, to efficiently dissipate the heat generated

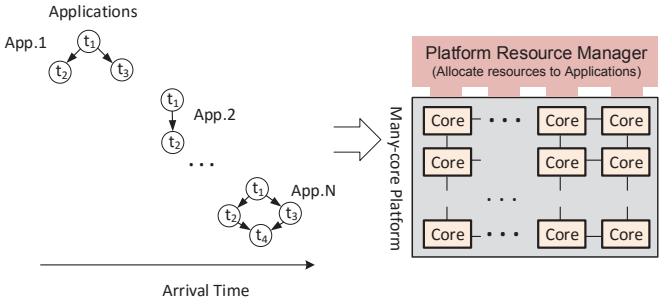


Fig. 2. System model.

by the active cores. However, static workload has been assumed, *i.e.*, a fixed set of applications are allocated at the same time and thus the number of active/dark cores are fixed. For open workloads, the number of active/dark cores will vary depending upon the arrival and departure of the applications. Further, applications containing dependent tasks are not considered and thus communication overhead between the active cores containing the communicating tasks is ignored. In such cases, the appropriate location of active cores not only help to optimize the peak temperature but also the communication time/performance. The approach in [23] leverages dark silicon to balance the temperature of active cores to provide higher power budget and better resource utilization, within a safe peak operating temperature. Dynamic workloads are considered and this approach aligns active cores along with dark cores that can evenly distribute heat dissipation across the chip. However, the distribution of active/dark cores amongst multiple applications at the same time is not considered, which might degrade overall system throughput. Our approach addresses above concerns by appropriately identifying the number, locations and voltage/frequency levels of active cores for the applications containing dependent tasks and arriving at different moments of time as a dynamic workload.

TABLE 1
Notations used in this paper

Variables for the models and problem formulation	
$G(Q, L)$	The NoC model, Q is the set of cores, and L is the set of links
$AG_i = (A_i, E_i)$	The task graph model for application i, A_i is the set of tasks, and E_i is the set of communications among the tasks
$M(\cdot)$	The task-to-core mapping function
$Tr(e)$	The time for data communication of the two tasks connected by edge e
$D(\cdot, \cdot)$	The Manhattan distance between two cores
$\omega(\cdot, \cdot)$	The traffic volume between two tasks
ET_i	The execution time of the application i
Γ	The set of all free cores in the system
B_i	The set of bubbles associated with application i
τ	The discrete time units
C	The thermal capacitance matrix of on-chip components
A_C	The thermal conductance matrix
$T(\tau)$	The temperature vector
$P(\tau)$	The power vector
$P_M(t_i)$, $P_M(x, y)$	The power capacity of core t_i whose coordinate is (x, y)
$T_{x,y}^{\tau+1}$	The temperature of a core located at (x, y) at time instance $\tau + 1$
f_j	The frequency of core j
σ_i	The response time of application i
A_i^{arrive}	The arrival time of application i
A_i^{finish}	The finishing time of application i
σ	The response time of running N applications within a given time
Variables for the proposed method	
bn	The number of bubbles in an application
WT_i	The waiting time of application i
r	The average percentage of bubble count
λ	The average application arrival rate
$RG(\mathcal{V}, \mathcal{A})$	The state graph for the Rollout algorithm
$J(v_{i,j}, d)$	The cost-to-go of each vertex $v_{i,j}$ in the state graph
T_H	Thermal thresold
s, d	Dummy source and destination vertices in the state graph
Experimental results	
ε	The error of the waiting time model

3 System Model and Problem Definition

Figure 2 shows our target system model. The system contains a many-core platform that executes a set of applications arriving at different moments of time. The applications are submitted to the platform resource manager that allocates resources to them. This section provides a brief overview of the platform, workload, thermal model, and thermal power capacity model along with the problem definition. The important notations used throughout the paper are summarized in Table 1.

3.1 Many-core Platform Model

The many-core platform contains a set of cores connected by an interconnection network, which is modeled as a 2D mesh network with bidirectional links. The right hand side of Figure 2 shows an example platform. Each core consists of a processing unit, a cache and a network interface. It is represented as a directed graph $G(Q, L)$, where Q is the set of cores and L represents the connections amongst the cores. The system resource management is done by a centralized platform resource manager. We consider that one core works as the resource manager and this core cannot be a bubble. Without loss of generality, the core in the left bottom corner is selected. However, any other core can also be selected as the manager. The manager keeps a table of the active/inactive status of the other cores and the status is updated when tasks are allocated or complete their execution on the cores. Every core can run at different voltage/frequency levels similar to the one supported in AMD Opteron. We assume a set of voltage/frequency levels for each core, where there is a fixed frequency for a voltage level.

3.2 Application Model

Each application i is represented as a directed graph $AG_i = (A_i, E_i)$, where A_i is the set of tasks of the application and E_i is the set of directed edges representing dependencies amongst the tasks. A task can also be referred to as a thread or process of an application. The left hand side of Figure 2 shows some example application graph models. The arrived applications are entered into First-In First-Out (FIFO) queue. When a new application arrives in the system, it is first placed in the system queue. The resource manager core checks the queue. If one application is found to be ready to execute, its program and data will be loaded to the selected cores to run. The manager keeps a table of the

active/inactive status of the cores and selects cores for each application by following the proposed algorithm. Once an application finishes execution and leaves the system, the manager is notified and sets the active/inactive status of the corresponding cores to be inactive. Each task $a \in A_i$ has a weight: execution time (ExecTime), when mapped onto a core. The ExecTime for each task is considered as its worst-case execution-time (WCET) and remains fixed at a given frequency. Each edge $e \in E_i$ represents data volume communicated between the dependent tasks.

A mapping function $M(a) = t$, for $a \in A_i, t \in Q$ binds tasks to the cores, such that task a is mapped to core t. Each edge $e \in E_i$ has a weight of transmission time, when the two communicating tasks are mapped. The transmission time between two tasks depends on the communication distance between the cores on whom they are mapped and the traffic volume. We assume one task per core model [6]. For each edge $e = (a_i, a_j)$, the transmission time $Tr(e) = f(\omega(a_i, a_j), D(M(a_i), M(a_j)))$, where $\omega(a_i, a_j)$ is the traffic volume between the two tasks a_i and a_j, and $D(M(a_i), M(a_j))$ is the distance (hop counts) between two cores on whom tasks a_i and a_j are mapped. The function $f(\cdot, \cdot)$ models the transmission time versus the traffic volume and the hop count distance of the two tasks, which can be found by a linear regression as follows.

$$Tr(e) = \alpha \cdot \omega(a_i, a_j) + \beta \cdot D(M(a_i), M(a_j)) \quad (1)$$

where α and β are regression coefficients. The transmission time model can be trained offline by transmitting packets to measure the latencies. The execution time of each application i is the makespan of task graph, denoted as ET_i.

The set of all existing free cores in the system is denoted as Γ. A set of bubbles $B_i = \{t_1, t_2, ...\}$ are also associated with application i, where $t_1, t_2, ...$ are powered off cores for cooling.

3.3 Thermal Power Capacity Model

We define the thermal power capacity (TPC) of a core as the maximum power the core can consume given the power distribution of other cores, such that the whole chip's maximum temperature and thermal gradient do not exceed their respective thresholds. The TPC of each core can be determined at offline. In the rest of the paper, we use $P_M(t_i)$ and $P_M(x, y)$ to denote the power capacity of the core t_i at the location (x, y) interchangeably.

The TPC of a core is bounded by the cooling capacity of the system, and the power consumption or temperature of other cores, *i.e.,* thermal correlation. The thermal correlation, indicating the inter-dependency of the temperature of different cores, can be modeled by a linear regression [20]. The temperature $T_{x,y}^{\pi+1}$ at time instance $tau + 1$ of a core located at (x, y) can be determined by the temperature values of those cores located at $(x \pm l, y \pm l)$ at time π [20],

$$T_{x,y}^{\pi+1} = \phi(T_{x\pm l, y\pm l}^{t}) \quad (2)$$

where $\phi(\cdot)$ is a linear function, and l can be $0, 1$, representing core (x, y)'s neighboring cores.

Similarly, the TPC of a core t_i can be found as,

$$P_M(x, y) = \theta(P(x \pm l_1, y \pm l_2)) + \beta_q \overline{P} = \sum_q \alpha_q \cdot P(t_i) + \beta_q \overline{P} \quad (3)$$

where $P(x \pm l_1, y \pm l_2)$ is the power consumption of the core n_q located at $(x \pm l_1, y \pm l_2)$, which is thermally correlated with t_i. The function $\theta(\cdot)$ can also be found by autoregressive model (AR), using the lasso method [16]. In particular, for each core at (x, y) we only keep the coefficients of adjacent cores as non-zero. That is, $(x \pm l_1, y \pm l_2)$ with l_1 and l_2 equal to $0, 1, 2$, *i.e.,* cores that are neighboring to the core (x, y). These cores have the highest thermal correlations with the core (x, y). We set the coefficients of other cores to be 0, for core i. $\overline{P}$ is the average power consumption of the other cores in the chip. β_q is the regression coefficient.

Since the purpose of TPC model is for changing the cores' V/F levels, we only consider the power related to the instruction execution. The power consumption of remote memory access and network, which is directly related to the communication distance, is optimized in the virtual mapping step in Section 4.4. The dynamic power of a core i P_i is determined by the following equation.

$$P_i = e_i \cdot \theta_i \cdot f_i \cdot V_i^2 \quad (4)$$

where f_i is the frequency of core t_i, e_i is the regression coefficients, and θ_i is the instruction throughput (IPC) of core t_i, V_i is the voltage. The IPC of each core can also be calculated from the many-core simulator. Therefore, with Eqn. 4, one can compute the dynamic power P_i given the IPC workload and frequency of core t_i. It is compared against the maximal allowed power $P_M(i)$ computed from the TPC model in Eqn. 3. If P_i is higher than $P_M(i)$, the frequency and voltage level of this core should be scaled down until P_is is not higher than $P_M(i)$.

The TPC model are trained at offline. Eqn.3 can be used at runtime to estimate the maximum power a core can run given the power consumption of its neighboring eight cores with low computing cost. This model is trained as follows.

- At offline,
  - Multiple test vectors including $tv_j =< P1, , P12, \overline{P} >$ are generated, where $P1, ..., P12$ denote the power consumptions of neighboring cores as in Fig. 3. $\overline{P}$ is the average power consumption of the remaining cores (the blank (white) cores in Fig. 3).
  - For each tv_j, the maximal power consumption $P_M(i)$ of core t_i can be calculated by iteratively increasing the power consumption of core t_i (setting its V/F level from the lowest level to the highest one) until its temperature is going to violate the threshold. In other words, each input vector tv_j generates an output $P_M(i)$.
  - After generating multiple inputs and outputs, we use the maximal likelihood methods to find the coefficients in Eqn. 3 and store them to be used at runtime.
- At runtime, the above coefficients are used to compute the TPC.

3.4 Problem Statement

Within a given time period, for N applications arriving at different moments of time, the objective is to minimize

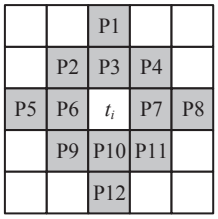


Fig. 3. The non-zero coefficients of Eqn. 3 corresponding to the neighboring cores.

the response time of each application in order to optimize throughput that is computed as the number of applications executed within a fixed amount of time. The decision variables are the position and number of the bubbles to be allocated to each application, together with the task-to-core mapping of each application. The response time of each application is computed as follows:

$$\sigma_i = A_i^{\text{finish}} - A_i^{\text{arrive}} \tag{5}$$

where, σ_i is the response time of application i, A_i^{arrive} and A_i^{finish} are respectively the arrival time and the finishing time of application i.

For each application, its response time is related to both the execution time and the waiting time. Waiting occurs when an application arrives at the system but there are no sufficient cores to run it. Execution time is related to both the communication and computation performances of the application.

The response time of running N applications within a given time is then computed as:

$$\sigma = A_{\text{finish}}^N \tag{6}$$

where N is the number of applications arrived at the system within a given time, and A_{finish}^N represents finishing time of N^{th} application within this given time.

The objective is to

$$\min \sigma \tag{7}$$

The constraint is that, each core is running with it's power consumption below the maximum power capacity, which is obtained from the TPC model in Eqns. 3.

The complexity of the problem can be analysed as follows. Suppose the many-core system has n cores and there are B bubbles (spare cores) at a certain time. If a total of m applications are to be executed in the system, in the worst case, there are B^m possible ways of assigning the bubbles to the applications. Once each application gets the bubbles, there is n^B possible ways of placing the bubbles. In summary, the worst case complexity of the problem is $O(B^m n^B)$, which indicates that the complexity increases with the number of applications and cores. Thus it is a NP-hard problem and motivates the need for an efficient heuristic.

4 Proposed Dynamic Resource Allocation Approach

4.1 Overview

Fig. 4 shows the overview of our proposed approach. Applications dynamically arrive in the system. The bubble count (number of bubbles) included in each application's core region (the region including active cores and bubbles) is used as a control variable, which determines both the communication distance and the running frequency of the active cores such that the system thermal constraint is not violated. Based on this, an application's region is defined as the cores that are dedicated to this application, including the cores running the tasks of the application and the bubbles budgeted to it. A *virtual mapping* process is first called to estimate the performance of each application when using different number of bubbles. For each application, core regions with different numbers of bubbles are selected, such that the region's core count is possibly larger than the number of tasks in the application. The tasks of the application are mapped virtually to this core region in order to estimate the performances given different bubble counts (b_n) for the application, *i.e.*, the table from the performance model achieved as shown in Fig. 4. The running frequency of each active core can be determined to confine to thermal constraint. During virtual mapping, no task is running on the cores, *i.e.*, the tasks are not actually mapped to the cores. The waiting time model also generates a table indicating the waiting time given different bubble counts. Finally, during the real or final mapping, the bubble count for each application is chosen which can result in the minimum application execution time (including communication and computation performances) and waiting time. Once the application finishes execution, the cores in the region is released and sent back to the available resource pool.

The reason to use the virtual mapping is as follows. The communication performance of each edge depends on the distance of the two cores running the communicating tasks, and the computation performance of each task depends on the frequency and TPC of each core, which is affected by the bubble count and location. Therefore, the calculation of execution time of an application requires knowing the task-to-core mapping scheme. To find the core region with the optimal number of bubbles, we need to consider both the execution time and waiting time of each application with different number of bubbles (the decision variable). Virtual mapping serves for this purpose. It iterates the bubble number $0, 1, ..., \min\{|A_i|, \Gamma\}$ for each application to generate the performance and waiting time models as shown in Fig. 4. The waiting time and performance models are stored in tables whose entries are $< bn, WT_i >$ and $< bn, ET_i >$, respectively. That is, given bn bubbles to be inserted into the application i, the two tables return the corresponding expected waiting time and execution time of the application. The mapping scheme with bn is also stored in a database. Based on the two models, during the final or real mapping, the system can choose the best bn value (bubble number) in the core region and the corresponding mapping scheme from the database for each incoming application, which can result in the minimal expected response time.

The various steps of the proposed approach are introduced in subsequent sub-sections and highlighted in Fig. 4.

4.2 Waiting Time Estimation

We target server systems whose workloads exhibit periodical behaviours [13], such that we can predict the waiting time from history data. In many server systems, there are some peak time when the CPU utilization is close to 100%,

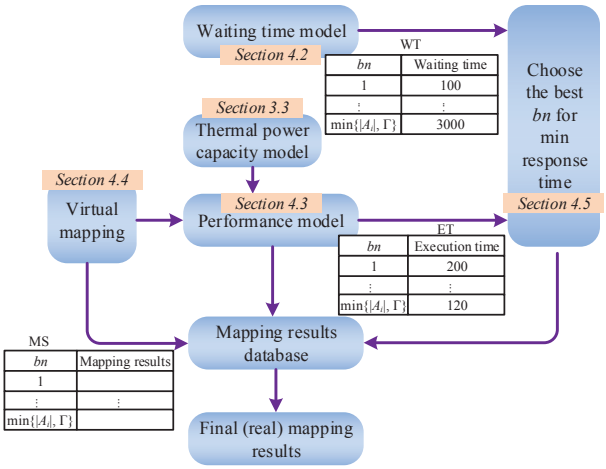


Fig. 4. Overview of the proposed approach.

and some off-peak time when there are fewer running applications. In addition, waiting time depends on various system parameters including the application arrival rate (average number of applications arriving in the system per cycle), system size, and how many free cores are used as bubbles.

The waiting time can thus be modeled by a polynomial regression model as in Eqn. (8), where $|Q|$ is the network size, $|A_i|$ is the average number of tasks in each application, r is the average percentage of bubble count in an application's core region, defined as bubble count divided by the core count in each application's core region, ET_i is the average execution time of the tasks, and λ is the average application arrival rate. Using this model, r can be a decision variable such that, when the waiting time is estimated to be high, a smaller r is preferred.

$$WT_i = \sum_{j=1}^{z} c_j \cdot |T|^j + \sum_{j=1}^{z} d_j \cdot |A_i|^j + \sum_{j=1}^{z} e_j \cdot r^j + \sum_{j=1}^{z} f_j \cdot {ET_i}^j + \sum_{j=1}^{z} g_j \cdot \lambda^j + a_0 \quad (8)$$

To find the coefficients of c, d, e, f, g, and a_0, the maximum likelihood methods can be used [16].

4.3 Performance Estimation

To estimate the performance of each application, we need know the communication performances of the edges and the computation performances of tasks in the task graph. These performances can only be determined after the tasks are mapped to cores. The number of bubbles in a core region is an important control variable which is related to both the communication distance and the computation power of each core/task. Given a virtual mapping of tasks to a core region with j bubbles, the execution time of each task and transmission time of each communication edge can be determined as in Sections 3.2 and 3.3. The execution time of each task is related to the instructions to be executed and the running frequency and power of the core while satisfying the thermal constraint, which can be derived from Section 3.3. The communication time of each edge in task graph can be determined by Eqn. 1. The performance of the application (referred to as makespan) can be determined by finding the maximum execution path along the application's task graph. Therefore, the performance estimation needs the virtual mapping algorithms which will be introduced in Section 4.4.

The output of the performance model as shown in Fig. 4 is a table `ET` where each item `ET[j]` is the execution time with j bubbles.

4.4 Virtual Mapping Algorithms

During the mapping process, we virtually find core regions whose core count equals to $|A_i|$ plus j bubbles, where $j = 0, 1, 2, ..., \min\{|A_i|, \Gamma\}$. At each iteration with j bubbles, the applications are virtually mapped to the core region and the execution time is stored in the performance model table. Once the iteration stops, the performance model generates a table indicating the execution times with j bubbles, where $j = 0, 1, 2, ..., \min\{|A_i|, \Gamma\}$. The corresponding mapping schemes with up to j bubbles are also stored in a database. Note that, this process only virtually maps the tasks to the cores to get the performance model table and the mapping scheme database as shown in Fig. 4. Tasks are not actually bound to and run on the cores. No migration is involved. Other running application is intact.

The virtual mapping process has two objectives, *i.e.*, minimizing the communication distance and maximizing the computation frequency/performance of the tasks. These two objectives might be contradicting in the sense that, communication distance is minimal when tasks are mapped in close proximity, while each task's frequency or computation performance is maximized when the temperature is low indicating hot tasks are distant from each other. We propose a heuristic based virtual mapping algorithm, where the two optimization objectives are tried to be achieved simultaneously.

Algorithm 1 shows the virtual mapping flow. At each iteration with j bubbles, the tasks are mapped to a core region of size $|A_i| + j$. The results are the two lookup tables `ET` and `MS`, where `ET[j]` returns the execution time and `MS[j]` returns the best virtual mapping scheme when inserting j bubbles, respectively.

Our proposed virtual mapping algorithm has the following steps.

1) Determine the computation to communication ratio (CCR), which is defined as the average computation workload (instructions to be executed) divided by the data volume to be sent in one application.
2) If CCR is over a threshold, call the computation biased virtual mapping sub-routine. Otherwise, call the communication biased virtual mapping sub-routine.

A larger CCR indicates each task computation performance contributes more to the overall application performance, while a small CCR means communication has more contribution to the application performance. Based on the CCR value, two virtual mapping sub-routines are called which are computation or communication biased. Both of the two mappings have two steps as follows. An initial mapping is set up first, followed by an iterative replacement procedure to optimize computation and communication performances. The inputs to both of the virtual mapping

ALGORITHM 1: Online Virtual Mapping

Input: j: The bubble number.
Output: `ET[j]`: The execution time when inserting j bubbles.
`MS[j]`: the best mapping scheme when inserting j bubbles.
Function: Find the best virtual mapping scheme and the execution time for an incoming application given the bubble number is j, where $0 \leq j \leq \min\{|A_i|, \Gamma\}$.
begin
  if $CCR < Threshold$ **then**
    Call the `Communication Biased Virtual Mapping Sub-routine`;
  end
  else
    Call the `Computation Biased Virtual Mapping Sub-routine`;
  end
end

sub-routines are 1) the task graph of the incoming application, 2) the available cores in the system, and 3) the bubble number j, where $j = 0, 1..., \min\{|A_i|, \Gamma\}$.

4.4.1 Communication Biased Virtual Mapping Sub-routine

Algorithm 2 shows the communication biased virtual mapping sub-routine.

4.4.1.1 Initial Mapping: In the initial mapping, the objective is set to be minimal communication distance. A convex core region is first found, followed by tasks with larger communication volume mapped in closer proximity virtually. The mapping algorithm in [14] is used as the initial mapping with minimal communication distance as the optimization objective.

4.4.1.2 Inserting Bubbles: In each iteration, j bubbles are virtually inserted into the core region of this application to boost the computation performance of certain tasks, where $j = 0, 1..., \min\{|A_i|, \Gamma\}$. The application's core region is bounded by a convex hull. At each iteration with j bubbles, first, a location (x_1, y_1) inside the current convex hull is found, then a location (x_2, y_2) outside the convex hull is found that is adjacent to its boundary, and has the minimum distance to (x_1, y_1). The bubble is virtually moved from (x_2, y_2) to (x_1, y_1) using the path migration algorithm in [31]. As an example, Fig. 5 shows the process of inserting two bubbles iteratively. At each iteration, when a new bubble is to be inserted, each task is selected as the candidate to be replaced by the bubble. A bubble with the minimal distance to each task is virtually replaced with the task. Then, the maximum power/thermal budget and frequencies of the cores running the tasks are updated following the thermal power capacity model. After determining the frequency of each core and the communication distance of each edge in task graph, the computation and communication performances are updated following the application model in Section 3. The task replacement with the minimal execution time is recorded. For example, in Fig. 5, in the first step to insert one bubble, suppose replacing task 1 with a bubble leads to the minimal execution time. So task 1 is moved to the location of the bubble. The region is enlarged each time a bubble is inserted.

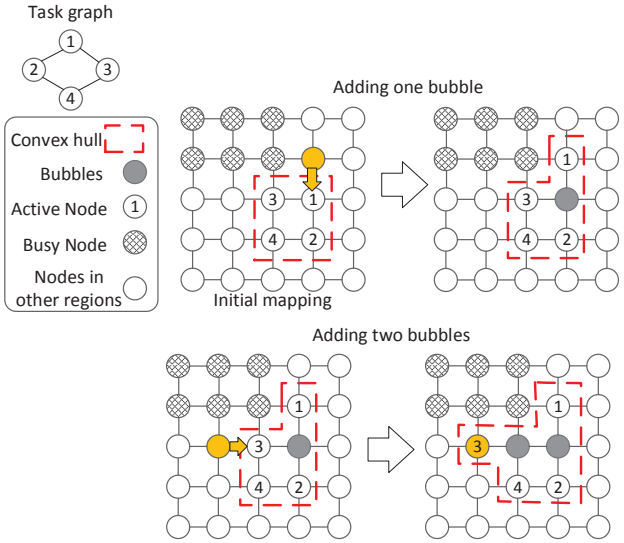


Fig. 5. Adding bubbles virtually to get the expected maximum speedup.

ALGORITHM 2: Communication Biased Virtual Mapping Sub-routine

Output: `ET[j]`: The execution time when inserting j bubbles.
`MS[j]`: the best mapping scheme when inserting j bubbles.
Function: Find the best mapping scheme and the execution time for an incoming application given the bubble number is j, where $0 \leq j \leq \min\{|A_i|, \Gamma\}$.
begin
  `/* Initial Mapping */`
  Map the tasks with communication-awareness by using [14] without bubble insertion;
  `ET[j] = INFINITY; // Recording the best performance`
  `/* Inserting Bubbles */`
  for $j = 0, ..., \min\{|A_i|, \Gamma\}$ **do**
    for *each active core* t_k *inside the core region* **do** `/* k = 0, 1, ..., min{|A_i|, Γ}, start with the hottest location */`
      Find a bubble b on the boundary of the core region returned by the mapping with the minimal distance to t_k;
      Virtually move b to t_k using [31];
      Update the performance `Ex`;
      if `Ex < ET[j]` **then**
        `ET[j] = Ex;`
        Virtually migrate b to t_k using [31] and update `MS[j]`;
      end
    end
  end
end

4.4.2 Computation Biased Virtual Mapping Sub-routine

Algorithm 3 shows the computation biased virtual mapping sub-routine.

4.4.2.1 Initial Mapping: If the task computation performance contributes more to the application performance, the initial mapping begins with a region of $\min\{2 \times |A_i|, \Gamma\}$ cores, where $|A_i|$ or Γ cores are powered off as bubbles. The tasks are sorted by their weight (each node's worst-case execution time in the task graph) in descending order. The

ALGORITHM 3: Computation Biased Virtual Mapping Sub-routine

Output: `ET[j]`: The execution time when inserting j bubbles.
`MS[j]`: the best mapping scheme when inserting j bubbles.
Function: Find the best mapping scheme and the execution time for an incoming application given the bubble number is j, where $1 \leq j \leq |A_i|$.
begin
  `/* Initial Mapping */`
  Find a core region with size of $\min\{2 \times |A_i|, \Gamma\}$;
  for *each unmapped task* a_k **do**
    Virtually map a_k to core t such that t has the maximum distance to other mapped tasks;
  end
  `ET[j] = INFINITY; // Recording the best performance`
  `/* Removing Bubbles */`
  for $j = 1, ..., |B_i|$ **do**
    for *edge* $e_k = (a_m, a_n)$ **do**
      Virtually move a_n to t_k, i.e., a core closest to $M(a_m)$ using [31];
      Update the performance `ET`;
      if `ET < ET[j]` **then**
        `ET[j] = ET;`
        Virtually migrate a_n to t_k using [31] and Update `MS[j];`
      end
    end
  end
end

tasks are mapped as distant as possible to each other. The mapping can be done as follows. For each unmapped task a_i in the sorted list, find a core t with maximal distance to the mapped tasks, *i.e.*, $\sum_{k=1}^{i-1} D(M(a_k), t)$. $D(M(a_k), t)$ is the distance of the core to a previous virtually mapped task. This equation finds the core that has the maximum distance to those running the virtually mapped tasks.

4.4.2.2 Removing Bubbles: To get the performances with different bubble counts for each application, the bubbles are virtually migrated out from the initial mapping region one by one at each iteration. The communication edges in the task are sorted by their volume in descending order. For each edge $e = (a_m, a_n)$, a_n is migrated to a free core virtually and the application performance is recalculated. If the performance is improved, a_n is virtually migrated to that free core and the bubble is migrated to the original location of a_n. Then, the bubble is excluded from the application. Fig. 6 shows two steps of virtually migrating task 3 towards task 1, and tasks 2 towards task 3. After virtually migrating one task to a bubble, the original core hosting the task is excluded from the region of this application. Then, the computation and communication performances are updated following the application model at each iteration.

4.4.3 Complexity Analysis

The worst-case complexity of the virtual mapping process can be analyzed as follows. In the communication biased virtual mapping algorithm, the initial mapping step has a complexity of $O(|A_i|^2 \cdot |E_i| \cdot K)$ [14], where $K = \max\{|A_i|, |\Gamma|\}$. In the second step, the algorithm has to iterate up to K times, corresponding to the bubble count.

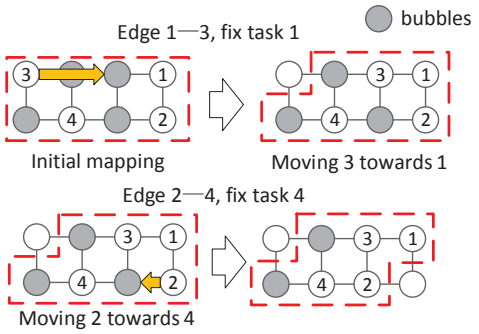


Fig. 6. Migrating bubbles virtually to optimize the communication distance.

For each bubble count j, it takes $O(|A_i|^2)$ steps to virtually migrate the tasks. In the computation biased virtual mapping algorithm, the initial mapping step has a complexity of $O(|A_i|^2 \cdot K)$, with $K = \max\{|A_i|, |\Gamma|\}$. In the second step, it also has to iterate up to $|A_i|$ times, corresponding to the bubble count. For each bubble count j, it takes $O(|E_i|)$ steps to virtually migrate the tasks. Overall, the worse case complexity is $O(|A_i|^2 \cdot |E_i| \cdot K)$, where $K = \max\{|A_i|, |\Gamma|\}$. For the other two approaches, DsRem [25] and PAT [23], their overhead or complexities are independent of the number of bubbles. However, the overhead of our proposed approach is comparable to that of PAT, and lower than that of DsRem.

4.5 Choosing the Best Number of Bubbles

Given the waiting time and the performance models versus bubble count, we can determine the number and locations of bubbles for each incoming application such that the overall system performance is optimized. To achieve the same, the following two steps are performed. First, using the above two models, we can select the number of bubbles $|B_i|$ for each application i with the minimum sum of execution time and waiting time, *i.e.*, $\min\{ET_i + WT_i\}$, with $0 \leq |B_i| \leq \min\{|A_i|, |\Gamma|\}$, where $|\Gamma|$ is the total number of free cores. Second, with a bubble count of $|B_i|$, the mapping results can be retrieved from the database `MS[`$|B_i|$`]` as shown in Fig. 4.

4.6 Power Budgeting for the Cores

Once the locations of the cores are determined by the above steps, the voltage/frequency (V/F) of the cores need to be set to appropriate levels so that they can run at a high speed without violating temperature constraint. One challenge in choosing the V/F levels of the cores is that, the TPC of one core depends on the heat generated from it's neighbor cores, and thus we cannot separately set each core's V/F level to achieve the optimal power budgeting scheme. In this section, we propose a Rollout algorithm based power budgeting method. The main idea of Rollout algorithm is as follows. In Rollout algorithm, a base heuristic algorithm is used first to set the V/F levels according to a simple rule (for example, set the V/F level of a core to a maximum that is allowable by it's TPC). To choose a particular V/F level of a core, the Rollout algorithm uses this base heuristic to "look ahead", i.e., estimate the possible application performance if this V/F level is chosen for the core. Fig. 7 shows the illustration of Rollout algorithm. At each step i (corresponding to choosing the V/F level of a core i), the decisions can be

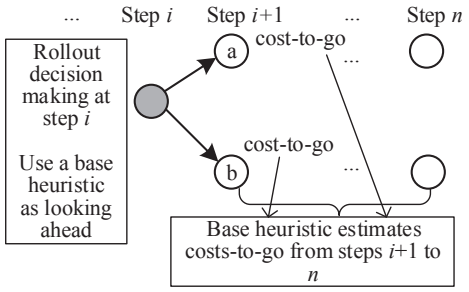


Fig. 7. Overview of the Rollout algorithm.

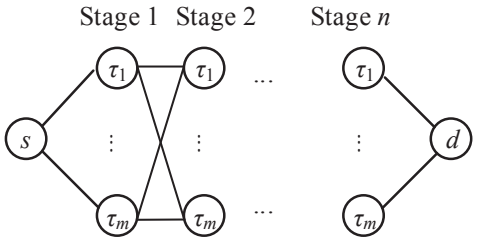


Fig. 8. The state space model of Rollout algorithm.

made to either choose a or b. The costs-to-go of choosing a and b are estimated using a base heuristic starting at a and b, respectively, and iterates toward the last step. The decision of Rollout at step i depends on the heuristic that iterates from steps $i+1$ to n.

In what follows, a state space model is first defined for the power budgeting problem, followed by the Rollout algorithm.

4.6.1 State space model for the power budgeting problem

For an application mapped to n active cores each of which can run at m V/F levels, an undirected graph $RG(\mathcal{V}, \mathcal{A})$ is defined as in Fig. 8.

1) The vertex set $\mathcal{V}$ has cardinality of $n \times m + 2$. Each column of m vertices form a stage, and there are n stages, corresponding to the n cores. Each vertex $v_{i,j} \in$ corresponds to core i running at V/F level j.
2) Two dummy vertices, s and d are added before the first stage and after the last stage. The edges connecting s to the vertices in stage 1 have a weight of 0.
3) Each vertex $v_{i,j}$ has a value $J(v_{i,j}, d)$ indicating the estimated cost-to-go returned from the base heuristic, corresponding to the execution time of setting core i's V/F level to be τ_j.

4.6.2 Base heuristic

The base heuristic is presented in Algorithm 4, which starts with an input vertex $v_{i,j}$ and iterates toward d. The policy is to choose the highest V/F level that is not violating the TPC of core k at each stage k. At stage q, the computation can be estimated as follows.

$$u_{q,x} = \arg \max_{\forall v_{q+1,y} \in \Pi_1} \{y\} \tag{9}$$

where Π_1 is a set of vertices in stage $q+1$ corresponding to the V/F levels such that the TPC of core q is below the thermal threshold T_H,

$$\Pi_1 : \{P_M(q+1, y) \leq T_H\} \tag{10}$$

ALGORITHM 4: Base Heuristic

Input: $v_{i,j}$: The starting vertex $v_{i,j}$.
Output: $v_{i+1,x}, ..., d$: The vertex selected at each stage from $i+1$ to n, corresponding to V/F level of each router.
Function: Find an edge connecting each vertex $v_{q,x}$ to a vertex at stage $x+1$, corresponding to the V/F level selection of the core q

```
begin
    for each stage q from i + 1 to d do        /* current
     vertex is v_{q,x} */
        for each edge (v_{q,x}, v_{q+1,y}) parallel do
            if P_M(q + 1, y) ≤ T_H then
                u_{q,x} = arg max_{∀v_{q+1,y} ∈ Π_1} {y} ;
        end
    end
end
```

After the base heuristic makes decision at each stages, a path from $v_{i,j}$ to d is obtained representing the V/F levels selection for cores $i, i+1, ...n$.

4.6.3 Rollout

The Rollout algorithm (Algorithm 5) improves the base heuristic iteratively from s to d. To make decision at stage i, it calls the base heuristic to estimate the costs-to-go of vertices from stages $i+1$ to d. Then it chooses the vertex with the best estimated cost-to-go at stage i.

At each vertex $v_{i,j}$, the V/F levels of cores $1, 2, ..., i-1$ are set by Rollout in previous steps. Decision needs to be made to choose among the vertices $v_{i+1,1}, ..., v_{i+1,m}$, corresponding to setting one of the m possible V/F levels for core i. For each choice $v_{i+1,t}$, Rollout calls the base heuristic to estimate the cost-to-go of choosing $v_{i+1,t}$, corresponding to tentatively setting the V/F levels of cores $i+1, ..., n$ by the base heuristic. With the V/F levels of all the cores to be known, the cost-to-go of $v_{i+1,t}$ can be computed which is the execution time of the task graph. The computation at each vertex $v_{i,j}$ can be given as,

$$u^*_{i+1,t} = \arg \min_{\forall v_{i+1,k} \in \Pi_2} \{ET\} \tag{11}$$

where ET is the execution time of the application after setting the V/F levels of cores $1, 2, ..., i-1$ by Rollout in previous steps, setting core i's V/F to be t, and tentatively setting the V/F levels of cores $i+1, ..., n$ by the base heuristic. Π_2 is a set of vertices in stage $i+1$ corresponding to the V/F levels that the TPC of core i is below the thermal threshold T_H.

$$\Pi_2 : \{P_M(i+1, k) \leq T_H\} \tag{12}$$

Fig. 9 shows an example of the Rollout algorithm at decision making for vertex $v_{2,2}$. The V/F levels of core 1 is already set at previous iteration. The next vertex can be $v_{3,1}$ or $v_{3,2}$. The base heuristic is called for both of the two vertices. The lower part of the figure shows the process of calling the base heuristic from $v_{3,2}$. The base heuristic selects the maximum V/F level under core 3's TPC. One the base heuristic finishes running, the V/F levels of cores 3 and 4 are tentatively set. The Rollout algorithm at $v_{2,2}$ knows the V/F levels of cores $1, 3, 4$ and makes decision for core 2 that results in the minimum application execution time.

ALGORITHM 5: Rollout Algorithm

Output: $v_{1,j}, ..., v_{n,j_n}$: The V/F level setting of each core.
Function: Find an edge connecting each vertex $v_{i,j}$ to a vertex at stage $i+1$, corresponding to the frequency assignment of the core i
begin
  for *each stage* i *from* s *to* j **do** `/* current vertex is` $v_{i,j}$ `*/`
    for *each edge* $(v_{i,j}, v_{i+1,k})$ **parallel do**
      if $P_M(i+1,k) \leq T_H$ **then**
        call the `Base Heuristic` routine in Algorithm 4 with input $v_{i+1,k}$
        $u^*_{i+1,t} = \arg\min_{\forall v_{i+1,k} \in \Pi_2}\{ET\}$;
    end
  end
end

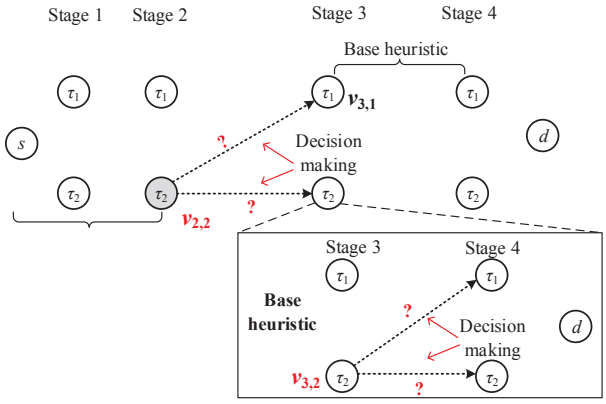


Fig. 9. An example of Rollout algorithm.

Assume each processor core has m V/F levels. Since the base heuristic traverses the $O(|Q|m)$ vertices and thus its complexity is $O(|Q|m)$. At each stage, the Rollout algorithm calls the base heuristic and there is a total of Q stages. Therefore, the overall complexity of the Rollout algorithm is $O(m|Q|^2)$.

5 Experimental Evaluation

5.1 Experimental Setup

We implemented a modified version of POPNET, which is an open source event-driven C++ network simulator [1], [34]. To simulate the temperature, we use an Alpha EV 6 like core floorplan, including L1 cache. The floorplan of each tile in NoC is shown as in Fig. 11, as in [41]. The dimension of the Alpha core is adopted from [42], which is scaled down to 45nm as in [22]. The area of a router can be obtained by running DSENT [39]. Table 2 lists the simulation configuration for Hotspot. We used discrete frequency levels from 1GHz to 3 GHz with 166MHz step size [26]. We use McPAT to compute the power consumptions of cores running the threads of each application. DSENT is integrated as the power model and Hotspot is used as the temperature simulator. Task graphs are modeled in this simulator, which can dynamically arrive at the system. The simulator system includes a network simulation subsystem which can model the package delay and energy of the communications. The configuration of the network-on-chip is listed in Table 3. The many-core system floorplanning can be found in [41]. The temperature threshold is 60 ^{o}C.

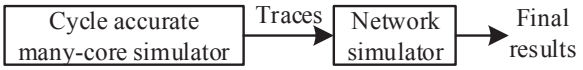


Fig. 10. Simulation flow.

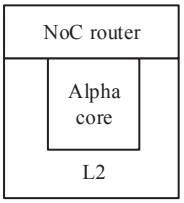


Fig. 11. The floorplan of each tile in NoC.

TABLE 2
The simulation configuration for Hotspot

	Thermal conductance (W/mK)	Heat capacity) (J/m^{3}K)	Depth (um)
Active layer	160.11	1.66×10^6	50
Interface layer	6.83	3.99×10^6	10
Heat sink	400	3.55×10^6	6900
C_{wire}(fF/mm)	212.12		
Ambient temperature	318K		

We compare our approach with the following two run-time thermal-aware mapping algorithms that aim to dark silicon era, *(1) DsRem* [25], where the cores on/off patterning are identified followed by tasks mapped to active cores, and *(2) PAT* [23], where a core region including inactive cores is found for each application. To augment existing algorithms, we assign a maximum of max $\{|A_i|, \Gamma\}$ bubbles to each application, where $|A_i|$ is the number of tasks of each application and Γ is the number of available bubble in the system.

Both random and real applications are used in the experiments as tabulated in Table 3 in order to evaluate the performance of the proposed and relevant algorithms considered for comparison. The task graphs of the real applications are generated from the traces of SPLASH-2 [3] and PARSEC [4]. These traces are collected by executing these applications in a 8×8 NoC-based cycle accurate many-core simulator. Then, a network simulator process the traces to achieve quick results for various system sizes. The network simulator models the package delay and energy of the communications based on the traces. The simulation flow is shown in 10. In particular, we compare throughput (defined as the average number of applications finished within a time unit), communication cost, and average waiting time for each application which occurs when there is insufficient cores to run the tasks that arrive in the system at run-time. The communication cost is defined as the network energy consumption, which is measured by DSENT. The run-time execution costs of the algorithms are also evaluated.

5.2 Testing Thermal Violation of Different TPC Models

TPC models in Eqn. 3 differ in time and space complexities. The more lightweight model in Eqn. 3 has lower time and space complexities, at the cost of ignoring the thermal impact of distant cores. Therefore, in this section, we perform a set of experiments to test the probability

TABLE 3
Simulation Configurations

Configuration of System Simulator for Extracting Traces	
Core Architecture	64 bit Alpha 21264
Baseline frequency	3 GHz
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
Network parameters	
Flit size	128 bits
Latency	Router 2 cycles, link 1 cycle
Buffer depth	4 flits
Routing algorithm	XY routing
Baseline topology	8×8
Random benchmark parameters	
Number of tasks	[15, 45]
Communication volume	[10, 200] (Kbits)
Degree of tasks	[1, 15]
Task number distribution	Bimodal, uniform
Task graphs of real applications	
barnes, blackscholes, fluidanimate, freqmine, ferret, vips, dedup, swaptions, canneal, streamcluster, raytrace	

of thermal violations (*i.e.*, the peak temperature is over the temperature threshold). The experiments are performed as follows. We perform N experiments using TPC models in both Eqn. 3 with each core's power consumption set randomly. We count the number of cases V_T that the peak temperature is over the threshold. We define the probability of thermal violation P_{VT} as,

$$P_{VT} = V_T/N \times 100\% \tag{13}$$

We run $108,800$ experiments, where in each experiment, the power consumptions of the cores are randomly set except a particular core t_i. The maximum allowed power (TPC) of core i $P_M(i)$ is computed by Eqn. 3. The maximum power consumption of core t_i together with the power consumptions of other cores are feed into Hotspot as input power trace to calculate the temperature. After performing $108,800$ experiments for TPC models in Eqn. 3, we find that using Eqn. 3 leads to no thermal violation in our experiments.

5.3 Validation of the Estimations

For errors in the waiting time estimation, Fig. 12 compares the linear regression and polynomial regression models. Fifty experiments are run with $|Q|$, $|A_i|$, r, ET_i and λ set randomly. The error of a single experiment is defined as,

$$\varepsilon = \frac{|WT - \widehat{WT}|}{WT} \times 100\% \tag{14}$$

where WT and $\widehat{WT}$ are the waiting times obtained from the simulator and the waiting time estimate model, respectively.

From this figure, one can see that quartic regression has the lowest error. Therefore, in the following experiments, we use the quartic regression model as the waiting time estimation. This indicates that the maximum order of the terms in Eqn. 8 is four.

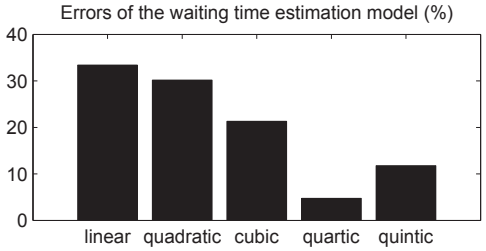


Fig. 12. Errors of different regression models.

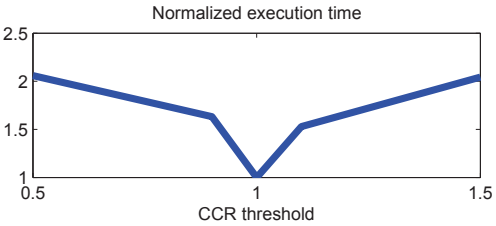


Fig. 13. CCR threshold selection.

5.4 Finding the CCR Threshold

Our approach (Algorithm 1) calls different sub-routines based on the CCR threshold and we have identified its value. Fig. 13 evaluates the CCR threshold which is used to classify an application as computation or communication biased. The communication volumes of the applications range from 20 to 1000 Kbits. CCR is defined as the sum of node weights divided by the sum of edge weights in each application's task graph. From this figure, one can see that, a CCR threshold of 1 generates the best performance. Therefore, in the following experiments, we set CCR threshold to be 1.

5.5 Performance Comparison

5.5.1 Evaluation on Random Benchmarks

Fig. 14 compares the throughput, waiting time, and communication cost at different network sizes, for the three methods. One can see that, when the network size is large, *e.g.*, 16×16, our approach can improve throughput by $1.7\times$ and $3.8\times$ over DsRem and PAT, respectively. The reason is that, our approach can optimize both the communication and computation intensive applications. For communication intensive applications, tasks with high traffic volumes are mapped closer, while for computation intensive applications, more bubbles are inserted. Therefore, our approach can achieve better performance. It can also be seen from Fig. 14 that the waiting time of our approach is shorter than the other two approaches because our approach balances the waiting time and the execution time of each application when inserting bubbles. The other two approaches only consider the performance of each individual application. Among the three approaches, DsRem has the worst communication cost, since it does not take the communications among the tasks into account.

Fig. 15 compares the considered metrics at different application communication volumes when the three methods are employed. It can be seen that when each application's average communication volume increases, *e.g.*, *150Kbits*, our approach's throughput is about $1.6\times$ and $1.8\times$ over DsRem and PAT, respectively. As DsRem does not consider communications among the tasks, it's performance gets worse

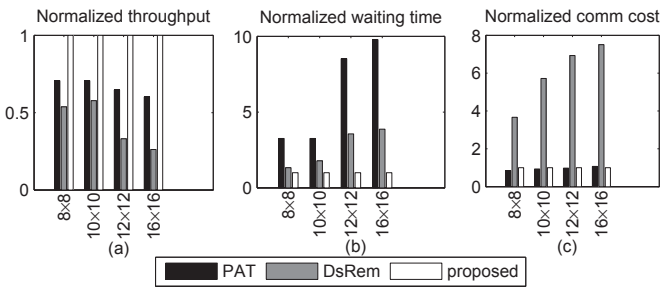


Fig. 14. The throughput, waiting time, and communication cost comparison at different network sizes.

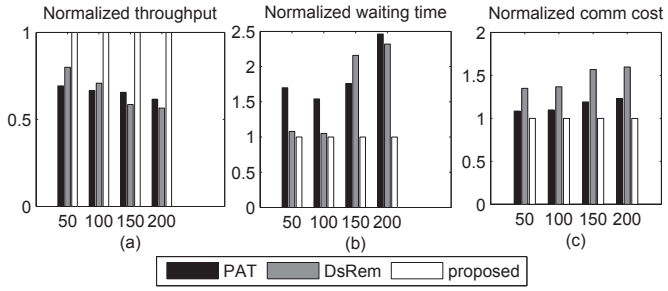


Fig. 15. The throughput, waiting time, and communication cost comparison at various communication volumes (in K bits).

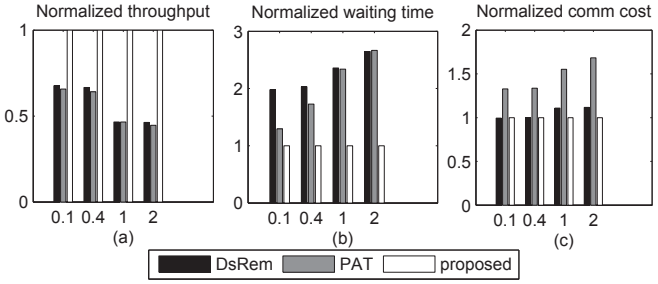


Fig. 16. The throughput, waiting time, and communication cost comparison at different application arrival rates (defined as the number of applications arrived per 100 cycles).

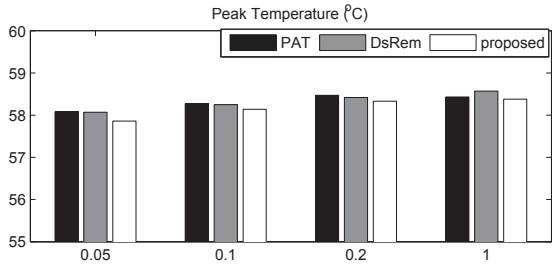


Fig. 17. The peak temperature comparison at different application arrival rates (defined as the number of applications arrived per 100 cycles).

when communication volume is large. Although PAT considers communication among the tasks, it does not consider budgeting the bubbles that affects the waiting time of future applications. Therefore, the waiting time of PAT is worse than ours, leading to a degraded throughput performance.

Fig. 16 compares the performances of the three methods with different application arrival rates. Application arrival rates is defined as the number of applications arrived at the system per 100 cycles, which measures the workloads of the system. When the arrival rate is high, *e.g.*, 2 applications arrive in the system per 100 cycles, our approach's throughput is about $2.16\times$ and $2.24\times$ over DsRem and PAT, respectively. A higher arrival rates means more applications arrive at the system, indicating the system workload is high. In such cases, DsRem and PAT might lead to long waiting time when applications arrive, since the free cores are used as coolers for currently running applications. For example, when the arrival rate is 2 applications per 100 cycles, the waiting times of PAT and DsRem are $2.64\times$ and $2.66\times$ of that of the proposed method. Further, DsRem and PAT optimize only for each individual application's performance. On the other hand, when the system workload is high, our approach budgets fewer bubbles to currently running applications and thus more free cores can be used to run the incoming applications, reducing their waiting time.

Fig. 17 compares the peak temperatures of the three methods. From Fig. 17, the peak temperature of the three approaches are close, and all of them are below the threshold of 60^oC.

5.5.2 Evaluation on Real Benchmarks

Fig. 18 compares the throughput, waiting time, and communication cost at different network sizes when the three methods are employed. When the network size is large, *e.g.*, 28×28, our approach's throughput is about $1.41\times$ and $1.42\times$ over DsRem and PAT, respectively. Our approach also reduces waiting time by 41% and 42% over DsRem and PAT, respectively.

Fig. 19 compares the considered metrics at different arrival rates for the three methods. When the arrival rate is high, *e.g.*, 2 applications arrive in the system per 100 cycles, our approach's throughput is about $1.4\times$ over DsRem and PAT, respectively. The reason is similar as in the case of random benchmarks.

5.6 Cost Analysis

The runtime cost of our algorithm is in the order of 1M cycles for the applications listed in table 3. This is averaged by running the algorithm fifty times with different system parameters. After the evaluation, it has been observed that the running times of PAT is also in the order of 1M cycles. DsRem, on the other hand, is designed for offline computation which takes longer time for computation. It has also been observed that the average application execution takes hundreds of millions of cycles in comparison to the 1M cycles for the algorithm runtime.

The runtime cost of the algorithm can be evaluated with respect to the average execution time of the applications. It has been observed that the average application execution takes 10^7 to 10^9 cycles in comparison to the 1M cycles of the algorithm runtime. Therefore, from the perspective of the application execution time, the overhead of the mapping algorithm is low.

The frequency of algorithm execution depends upon the workload arrival rate, *i.e.* it is applied as soon as an application arrives into the system. The arrival rate also defined the average time between two applications arrival. Based on the algorithm overhead, we recommend that our algorithm is suitable for systems with arrival interval larger than 10M cycles, which is 0.03 sec with CPU clock frequency of 3 GHz.

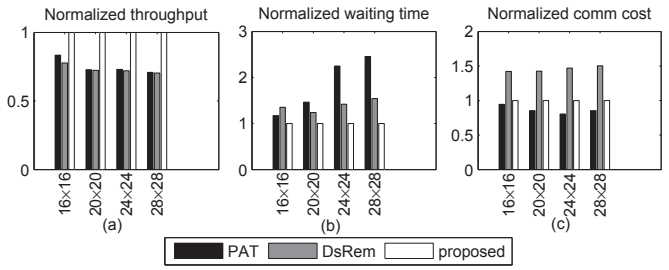


Fig. 18. The peak temperature comparison at different arrival rates.

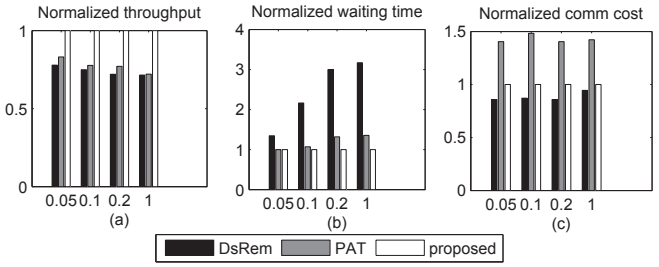


Fig. 19. The throughput, waiting time, and communication cost comparison at various application arrival rates.

6 Conclusion

We proposed an online algorithm to budget free cores (referred as bubbles) to each application so as to optimize the system throughput. The system throughput is related to each application's communication and computation performances, as well as the waiting time incurred when it finds insufficient cores to run its tasks. Performance and waiting time models are first set up for the applications. An online algorithm was proposed to find the best number and locations of the bubbles to each application, according to whether the new application is computation or communication intensive. The algorithm also trades the execution performance of each running application with the waiting time of new applications. A Rollout algorithm is proposed to budget power to the cores by setting the cores' V/F levels. Our experiments confirmed that, compared with two existing runtime resource management approaches, our approach can improve the system throughput by as much as 50%. The runtime overhead of our approach is moderate, making it a suitable runtime resource management approach to achieve high system throughput for many-core systems running open workloads.

References

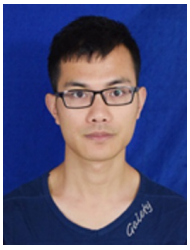
[1] Popnet, https://github.com/karellincoln/popnet.git.
[2] I. Anagnostopoulos, V. Tsoutsouras, A. Bartzas, and D. Soudris. Distributed run-time resource management for malleable applications on many-core platforms. In *Proc. Design Automation Conf.*, pages 1–6, 2013.
[3] J. M. Arnold, D. A. Buell, and E. G. Davis. Splash 2. In *In Proc. Int'l Conf. SPAA*, pages 316–322, 1992.
[4] R. Bagrodia, R. Meyer, M. Takai, Y.-A. Chen, X. Zeng, J. Martin, and H. Y. Song. Parsec: a parallel simulation environment for complex systems. *IEEE Trans. Computer*, 31(10):77–85, 1998.
[5] D. Bertsekas. Rollout algorithms for constrained dynamic programming. *Lab. for Information and Decision Systems Report*, 2646, 2005.
[6] S. Boyd-Wickizer, H. Chen, R. Chen, Y. Mao, M. F. Kaashoek, R. Morris, A. Pesterev, L. Stein, M. Wu, Y.-h. Dai, et al. Corey: An operating system for many cores. In *OSDI*, volume 8, pages 43–57, 2008.
[7] E. L. d. S. Carvalho, N. L. V. Calazans, and F. G. Moraes. Dynamic Task Mapping for MPSoCs. *IEEE Design Test*, 27(5):26–35, 2010.
[8] J. Castrillon, A. Tretter, R. Leupers, and G. Ascheid. Communication-aware mapping of kpn applications onto heterogeneous mpsocs. In *the 49th Annual Design Automation Conference*, pages 1266–1271, 2012.
[9] C.-L. Chou, U. Y. Ogras, and R. Marculescu. Energy-and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 27(10):1866–1879, 2008.
[10] A. K. Coskun, T. S. Rosing, K. A. Whisnant, and K. C. Gross. Temperature-aware mpsoc scheduling for reducing hot spots and gradients. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 49–54. IEEE Computer Society Press, 2008.
[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.*, pages 1–6, 2013.
[12] D. G. Feitelson and L. Rudolph. Metrics and benchmarking for parallel job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–24, 1998.
[13] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Pro. Int'l Symp. Workload Characterization*, pages 171–180, 2007.
[14] M.-H. Haghbayan, A. Kanduri, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen. Mappro: proactive runtime mapping for dynamic workloads by quantifying ripple effect of applications on networks-on-chip. In *Proc. Int'l Symp. Networks-on-Chip*, page 26, 2015.
[15] V. Hanumaiah, S. Vrudhula, and K. S. Chatha. Performance optimal online dvfs and task migration techniques for thermally constrained multi-core processors. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 30(11):1677–1690, 2011.
[16] T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The elements of statistical learning*. Springer, 2009.
[17] J. Henkel, H. Khdr, S. Pagani, and M. Shafique. New trends in dark silicon. In *Proc. Design Automation Conf.*, pages 1–6, 2015.
[18] W. Huang, M. R. Stant, K. Sankaranarayanan, R. J. Ribando, and K. Skadron. Many-core design from a thermal perspective. In *Proc. Design Automation Conf.*, pages 746–749, 2008.
[19] D.-C. Juan, S. Garg, J. Park, and D. Marculescu. Learning the optimal operating point for many-core systems with extended range voltage/frequency scaling. In *Proc. Int'l Conf. Hardware/Software Codesign and System Synthesis*, pages 1–10, 2013.
[20] D.-C. Juan, H. Zhou, D. Marculescu, and X. Li. A learning-based autoregressive model for fast transient thermal analysis of chip-multiprocessors. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 597–602, 2012.
[21] H. Jung, C. Lee, S.-H. Kang, S. Kim, H. Oh, and S. Ha. Dynamic behavior specification and dynamic mapping for real-time embedded systems: Hopes approach. *ACM Trans. Embed. Comput. Syst.*, 13(4s):135, 2014.
[22] M. Kadin. Frequency planning for multi-core processors under thermal constraints. In *Proc. Int'l Symp. Low Power Electronics & Design*, pages 213–216, 2008.
[23] A. Kanduri, M.-H. Haghbayan, A.-M. Rahmani, P. Liljeberg, A. Jantsch, and H. Tenhunen. Dark silicon aware runtime mapping for many-core systems: a patterning approach. In *Proc. IEEE Int'l Conf. Computer Design*, 2015.
[24] S. Kaushik, A. Singh, W. Jigang, and T. Srikanthan. Run-time computation and communication aware mapping heuristic for NoC-based heterogeneous mpsoc platforms. In *Proc. Int'l Symp. Parallel Architectures*, pages 203–207, 2011.
[25] H. Khdr, S. Pagani, M. Shafique, and J. Henkel. Thermal constrained resource management for mixed ilp-tlp workloads in dark silicon chips. In *Proc. Design Automation Conf.*, pages 1–6, 2015.
[26] S. Kim, J. Lee, and C. Kyung. 3D-stacked L2 cache configuration for DVFS-enabled processor to minimize overall energy consumption. In *Int'l Conf. Convergence and Hybrid Information Technology*, pages 1–4, 2010.
[27] P. Kumar and L. Thiele. Thermally optimal stop-go scheduling of task graphs with real-time constraints. In *Proc. Asia and South Pacific Design Automation Conf.*, pages 123–128. IEEE Press, 2011.
[28] J. Lee and N. S. Kim. Optimizing throughput of power-and thermal-constrained multicore processors using DVFS and per-core power-gating. In *Proc. Design Automation Conf.*, pages 47–50. IEEE, 2009.
[29] P. Marwedel, J. Teich, G. Kouveli, I. Bacivarov, L. Thiele, S. Ha, C. Lee, Q. Xu, and L. Huang. Mapping of applications to MPSoCs.

In *the IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 109–118, 2011.
[30] T. S. Muthukaruppan, M. Pricopi, V. Venkataramani, T. Mitra, and S. Vishin. Hierarchical power management for asymmetric multi-core in dark silicon era. In *Proc. Design Automation Conf.*, pages 1–9, 2013.
[31] J. Ng, X. Wang, A. Singh, and T. Mak. DeFrag: defragmentation for efficient runtime resource allocation in NoC-based many-core systems. In *Prpc. Euromicro Int'l Conf. Parallel, Distributed and Network-Based Processing*, 2015.
[32] S. Pagani, H. Khdr, W. Munawar, J. J. Chen, M. Shafique, M. Li, and J. Henkel. TSP: thermal safe power: efficient power budgeting for many-core systems in dark silicon. In *Proc. Int'l Conf. Hardware/software Codesign and System Synthesis*, pages 10:1–10, 2014.
[33] E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano. Customization of OpenCL applications for efficient task mapping under heterogeneous platform constraints. In *Proc. Design, Automation and Test in Europe*, pages 736–741, 2015.
[34] L. Shang, L. S. Peh, and N. K. Jha. Dynamic voltage scaling with links for power optimization of interconnection networks. In *Proc. Int'l Symp. High-Performance Computer Architecture*, pages 91–102, 2003.
[35] A. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: Survey of current and emerging trends. In *Proc. Design Automation Conf.*, pages 1:1–1:10, 2013.
[36] A. K. Singh, P. Dziurzanski, H. R. Mendis, and L. S. Indrusiak. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. *ACM Computing Surveys (CSUR)*, 50(2):24, 2017.
[37] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang. Communication-aware heuristics for run-time task mapping on NoC-based MPSoC platforms. *J. Syst. Archit.*, 56:242–255, 2010.
[38] T. Somu Muthukaruppan, A. Pathania, and T. Mitra. Price theory based power management for heterogeneous multi-cores. *ACM SIGARCH Computer Architecture News*, 42(1):161–176, 2014.
[39] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic. DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling. In *Proc. Int'l Conf. NoCS*, pages 201–210, 2012.
[40] 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*, pages 1–8, 2016.
[41] A. Y. Yamamoto and C. Ababei. Unified reliability estimation and management of NoC based chip multiprocessors. *Microprocessors & Microsystems*, 38(1):53C63, 2014.
[42] C. Zhu, Z. Gu, L. Shang, R. Dick, and R. Joseph. Three-dimensional chip-multiprocessor run-time thermal management. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1479–1492, 2008.

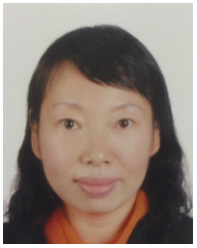
Xiaohang Wang received the B.Eng. and Ph.D degree in communication and electronic engineering from Zhejiang University, in 2006 and 2011. He is currently an associate professor at South China University of Technology. He was the receipt of PDP 2015 and VLSI-SoC 2014 Best Paper Awards. His research interests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

Amit Kumar Singh (M09) 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 and at University of York, UK from 2014 to 2016. Currently, he is working as senior research fellow at University of Southampton, 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 50 papers in the above areas in leading international journals/conferences. Dr. Singh was the receipt of 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 DATE, ISED, MES, NoCArc and ESTIMedia.

Bing Li received the bachelor degree in software engineering from South China University of Technology(SCUT), Guangzhou, China. She is pursuing her master degree in the department of software engineering, SCUT. Her research interest is task mapping for NoC-based systems.



Yang Yang received the bachelor degree from the School of Data and Computer Science, Sun Yat-sen University, China. Currently, he is pursuing master degree in the School of Data and Computer Science, Sun Yat-sen University. His research interest is mapping of applications on large scale multi-core architectures.



Hong Li is currently an associate professor at South China University of Technology. His research interests include many-core architecture, power efficient architectures, optimal control, and NoC-based systems.

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.