Runtime Mapping for Lifetime Reliability and Throughput Optimization in Many-Core Systems

Liang Wang, Xiaohang Wang, *Member, IEEE*, Ho-fung Leung, *Senior member, IEEE*, Terrence Mak, *Member, IEEE*,Jie Han *Member, IEEE* and Leibo Liu

***Abstract*—Due to technology scaling and increasing power density, lifetime reliability is becoming one of major design constraints in the design of future many-core systems. In this paper, we propose a runtime task mapping scheme which can dynamically map applications given a lifetime reliability constraint. A *borrowing strategy* is adopted to satisfy the lifetime reliability constraint in a long-term scale, while the lifetime constraint can be relaxed in local time scale when the communication requirement is high. The applications are classified into communication-intensive applications and computation-intensive applications, which are mapped using different strategies. The threshold to classify the applications is determined dynamically according to current locations of all available cores besides communication to computation ratio of applications. The proposed runtime mapping scheme can improve the throughput because the communication distance of communication intensive applications is optimized and meanwhile the waiting time of computation intensive application is reduced. The experimental results show that compared to the state-of-the-art lifetime-constrained mapping, the proposed mapping schemes can have up to 50.6% throughput improvement with only a small penalty on the communication cost.**

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, template.**

I. Introduction

Aggressive technology scaling has enabled integration of tens or hundreds of cores on a single chip [1], [2], which is known as many-core processor. The many-core systems provide high performance computing services for servers, data-centers, clusters, etc. The workloads on such a system usually exhibit dynamic feature, i.e. applications with difference characteristics arrive and leave the system dynamically. This necessitates an efficient runtime application mapping scheme, which can provide high throughput for the many-core system while satisfying system constraints such as limited resources, thermal design power (TDP), lifetime reliability, etc.

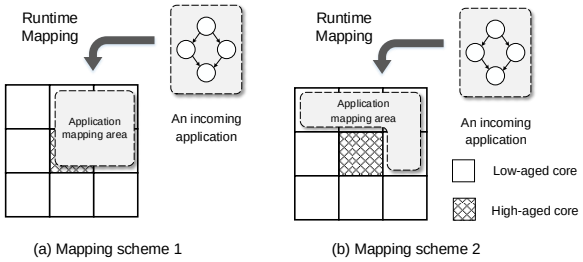


Fig. 1. Motivation example. (a) The application is mapped to a square region. (b) The application is mapped to a non-square region.

Because of technology scaling, lifetime reliability is becoming one of major design challenges for many-core system [3], [4], [5], [6]. The transistors are affected by various aging mechanisms such as Electro-migration (EM), Time Dependent Dielectric Breakdown (TDDB), Negative Bias Temperature Instability (NBTI), Hot Carrier Injection (HCI), etc [7], [8], which can eventually result from permanent failures of transistors. Mean-time-to-failure (MTTF), a widely used metric for lifetime reliability, is used to describe the expected lifetime of transistors [7]. To address the aging issues of many-core systems, various techniques such as dynamic voltage and frequency scaling (DVFS) [9], [3], [6], task mapping [10], [11], [12], etc. have been proposed. In addition, some studies have also taken the lifetime reliability as a design constraint [13], [14], [12] such that the performance is optimized while meeting a target MTTF. Existing state-of-the-art lifetime-constrained runtime task mapping algorithm [12] tries to map the applications to near-square regions in order to reduce the communication overhead and external contentions [15]. However, we notice that it is possible to lead to low throughput if the applications are always mapped to near-square regions. Fig. 1 presents an example, in which an application with 4 tasks is mapped to a 3×3 many-core system[1]. In the many-core system, the core in the central region is a high-aged core while others are less-aged. If the high-aged core has violated the pre-defined lifetime constraint, the incoming application has to wait until the core recovers from high-aged status. In Fig. 1(a), the application is mapped to a square region for lower communication overhead with the high-aged core selected. This would possibly lead to low

Liang Wang is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong and the with the Institute of Microelectronics, Tsinghua University, Beijing, China. Email: lwang@link.cuhk.edu.hk

Xiaohang Wang is with the School of Software Engineering, South China University of Technology, and Guangzhou Institute of Advanced Technology, Guangzhou, China. Email: xiaohangwang@scut.edu.cn

Ho-fung Leung is with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, NT, Hong Kong. Email: lhf@cuhk.edu.hk

Terrence Mak is with the School of Electronics and Computer Science, University of Southampton, Southampton SO17 1BJ, United Kingdom. Email: tmak@ecs.soton.ac.uk

Jie Han is with the Department of Electrical and Computer Engineering, University of Alberta, Canada. Email: jhan8@ualberta.ca

Leibo Liu is with the Institute of Microelectronics, Tsinghua University, Beijing, China. Email: liulb@tsinghua.edu.cn

[1]It is assumed that the maximum number of tasks on each core is 1.

throughput due to long application waiting time. However, the applications can have various communication characteristics, i.e. some applications are more computation-intensive. For those applications, it is not necessary to map to near-square regions as the communication distance is less important. As shown in Fig. 1(b), the application is mapped to a non-square region without occupying the high-aged core. Despite of slight increase of the average communication distance, the system throughput can be improved because of shorter waiting time for the recovery of high-aged core.

Another important feature of dynamic lifetime reliability (DRM) management is *borrowing strategy*, which is widely used in lifetime-constraint designs [9], [16], [17]. *Borrowing strategy* indicates that the lifetime constraint can be locally relaxed for better performance in local time window while still meeting the constraint over the large time window. However, most prior studies exploit *borrowing strategy* for DRM by DVFS in single-core or multi/many-core processors. And *borrowing strategy* is less exploited to manage lifetime reliability by runtime task mapping.

In this paper, we propose a runtime task mapping scheme with lifetime reliability as constraint. The runtime task mapping scheme aim to improve the global throughput of the system given a lifetime reliability constraint. The *borrowing strategy* is adopted to meet the lifetime constraint in a long-term scale. In other words, if the incoming application is a communication intensive applications, it is mapped to a near-square area with lifetime constraint locally relaxed because of higher communication performance requirement in local time window. Meanwhile, the computation intensive applications are mapped to an area with higher dispersion to free the high-aged cores. The global throughput is improved because the communication performance of communication intensive applications is optimized, and meanwhile the waiting time of computation intensive application is reduced. A preliminary work has been presented in [18]. This paper extends the preliminary work by adding substantial analyses, a novel method to classify applications and more experimental results including evaluations on realistic applications. The main contributions of this paper include:

(1) Propose a lifetime-constraint runtime task mapping scheme which exploits *borrowing strategy* to improve the system throughput by mapping computation intensive and communication intensive application with different strategies.

(2) Propose an improved neighborhood allocation scheme for runtime mapping.

(3) Propose a novel method to classify the applications into computation intensive and communication intensive applications.

The remainder of the paper is organized as follows. Section II briefly introduces the related work. Section III discusses the models of lifetime reliability and applications. Section IV presents the problem statement. Section V proposes the runtime mapping. Section VI analyzes the experimental results and Section VII concludes this paper.

II. Related Work

Lifetime-aware task mapping problems, which incorporate MTTF as an optimization objective or a constraint, have been solved with both design-time and runtime solutions.

A. Design-time Solutions

Huang et al. [10] define an aging effect for multi-core system, and propose a task mapping problem to maximize MTTF of multi-core system given performance constraints. In [13], Huang et al. propose another task mapping problem which minimizes the energy consumption under a lifetime constraint on multi-core system. Both the problems in [10], [13] are solved by simulate annealing (SA) at design-time. Meyer et al. [19] propose a slack allocation for performance and cost-constrained applications. By a design-time task mapping, the lifetime and manufacturing cost are jointly optimized. Zhu et al. [20] exploit task mapping, processor-level structural redundancy, floorplan, etc. to improve the MTTF of multi-core system given functionality and timing constraints. All these approaches address the lifetime reliability issues through design-time mapping.

B. Runtime Solutions

Hartman et al. [21] propose a lifetime-aware runtime mapping heuristics for NoC-based chip multiprocessors. In their heuristics, the cores with smaller accumulated aging effects are favored during the mapping process. Thus the heuristics minimizes the accumulated aging effects of the cores and extends the lifetime. Rathore et al. [22] consider process variation and propose a design-time/runtime hybrid method to optimize lifetime of the many-core system. At design-time, a simulated annealing algorithm is adopted to find a pareto-optimal mapping. At runtime, the tasks are migrated to healthier cores pro-actively. Sun et al. [23] propose a workload balancing scheme for multi-core systems to relax NBTI stressed cores. First, a capacity rate is defined for each core according to NBTI as an indication to accept workload. Then, a dynamic zoning algorithm is proposed to group cores into zone and a dynamic task scheduling algorithm is proposed to allocate tasks into each zone with balanced workload. The work extends 30% MTTF for multicore systems. All these methods target at maximizing MTTF of system. Some other researchers take the lifetime as a constraint to meet target MTTF at runtime. Gnad et al. [14] propose a run-time lifetime management system which harnesses dark silicon to decelerate aging, and improves the overall system performance for a given lifetime. However, their method targets at malleable application which has a varying degree of parallelism, thus not applicable for task graph based applications. The method in [12] extends the runtime mapping in [24] to take the aging information into consideration when selecting the first node. Based on the first node selection method, a runtime mapping given lifetime constraint is proposed. However, the method does not exploit the *borrowing strategy* feature of DRM. In other words, it is neglected that the lifetime constraint can be temporally locally relaxed for better performance requirement.

In this paper, some applications are mapped in dispersed regions to improve throughput. Fattah et al. [25] propose a similar approach, in which a parameter α named *allowed level of dispersion* is defined. They show that when $0 < \alpha < 1$, higher throughput can be achieved because of less waiting time. However, the *allowed level of dispersion* keeps constant for all applications.

Different from the approach in [25], we propose a hybrid mapping which classifies the applications into communication intensive applications and computation intensive applications. The computation intensive applications are allowed to map in dispersed regions while the communication intensive applications can only be mapped in compact regions. Meanwhile, the lifetime-constrained runtime mapping incorporates the *borrowing strategy* [9] to optimize the communication distance as the first priority in local time scale, and manages the lifetime in a long-term scale.

III. Models

A. Lifetime Reliability

The aging mechanisms for lifetime reliability include Electro-migration (EM), time-dependent dielectric breakdown (TDDB), stress migration (SM), negative bias temperature instability (NBTI), hot carrier injection (HCI), etc. In this paper, we only consider EM-induced aging mechanism. Other aging mechanisms can be easily incorporated using Sum-of-Failure-Rate (SOFR) models [13].

The lifetime reliability can be modeled according to Weibull distribution [10]. The MTTF is given as

$$MTTF = \alpha\Gamma(1 + \frac{1}{\beta}) \tag{1}$$

where α is related to operating conditions such as temperature, switching activity, frequency, etc. Γ is the gamma function. β is the slope parameter for Weibull distribution. Considering the runtime variation of operating conditions, an aging effect for a processor is also defined in [10], shown in Eq. 2.

$$A = \sum \frac{\Delta t_i}{\alpha_i} \tag{2}$$

where Δt_i is the duration of the i-th time interval. α_i is the scale parameter in the i-th time interval. The aging effect integrates all operating conditions including temperature, voltage, frequency and utilization into a single value.

To satisfy the pre-defined lifetime constraint, the aging effect should be less that the target aging effect. We define a lifetime budget in Eq. 3.

$$\Delta A(t) = A_{target}(t) - A(t) > 0 \tag{3}$$

A positive lifetime budget value means the lifetime constraint is satisfied and vice versa. For lifetime reliability management, the lifetime can be regarded as a resource consumed over time. $A_{target}(t)$ indicates the given resource, and $A(t)$ indicates the consumed resource. As soon as the consumed resource is less than the given resource, the lifetime constraint is satisfied. Eq. 2 and Eq. 3 also indicate that if a healthy core is over stressed for a period, the lifetime budget decreases and even the lifetime constraint is violated. On the contrary, if a high-aged core ($\Delta A < 0$) is less stressed for a period, it can recover from high-aged status.

B. Application Model

Each application can be modeled as a directed task graph $AG = (T, E)$, where T is the set of tasks, and E is the set of directed edges. Each task $t_i \in T$ has a worst-case execution time when mapped to a core. Each edge $e_{i,j} \in E$ represents data dependency between the source task t_i and destination task t_j. The weight $w_{i,j}$ represents the communication volume from task t_i to task t_j. The execution time of an application or the makespan of a task graph is determined by both task execution time and communication time.

C. Platform Description

The many-core system consists of a set of homogeneous cores which are connected by a 2D-mesh on-chip interconnect network. The platform is represented as a directed graph $G(P, L)$, where P represents the set of cores n_i and $l_{i,j} \in L$ represents the physical channel between two cores n_i and n_j. Each core is associated with a ΔA value. The resource management manager maintains an aging map, which contains the ΔA values of all cores. If the size of the many-core system is $M \times N$, the aging map can be represented by a matrix in Eq. 4.

$$\boldsymbol{AM} = \begin{Bmatrix} \Delta A_{1,1} & \cdots & \Delta A_{1,N} \\ \vdots & \ddots & \vdots \\ \Delta A_{M,1} & \cdots & \Delta A_{M,N} \end{Bmatrix} \tag{4}$$

D. Overview of the Proposed Methodology

A task mapping defines how an application is mapped on a set of cores. It is assumed the maximum number of tasks running on a core is 1. The mapping function $map(t_i)$ defines which core that the task t_i is mapped on. The application mapping and resource management are controlled by a centralized resource manager. Fig. 2 shows the diagram of the proposed lifetime-constrained mapping. Fig. 2(a) shows the applications which are represented by task graphs. In Fig. 2(b), an aging map is used to keep the lifetime budgets ΔA of the cores in the many-core system. Fig. 2(c) shows the many cores which are interconnected by on-chip network. At runtime, the resource manager takes the application information and platform aging map as the inputs of the mapping algorithm. The output of the algorithm specifies how the tasks of the incoming applications are mapped to the many-core system.

IV. Problem Statement

At runtime, there are N applications arriving at the many-core system within a given time period. To optimize the global throughput, the objective is to minimize the total makespan of all applications. The decision variable is the task-to-core mapping of each application. The makespan of each application is represented as

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

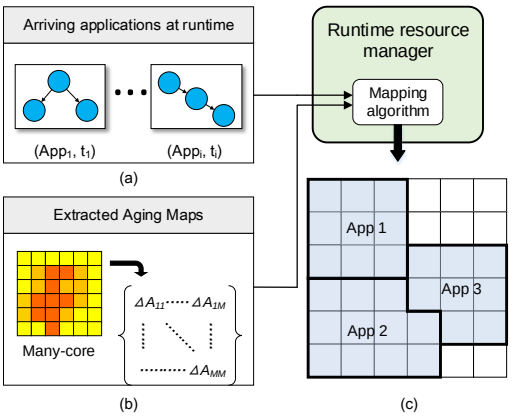


Fig. 2. Task mapping overview

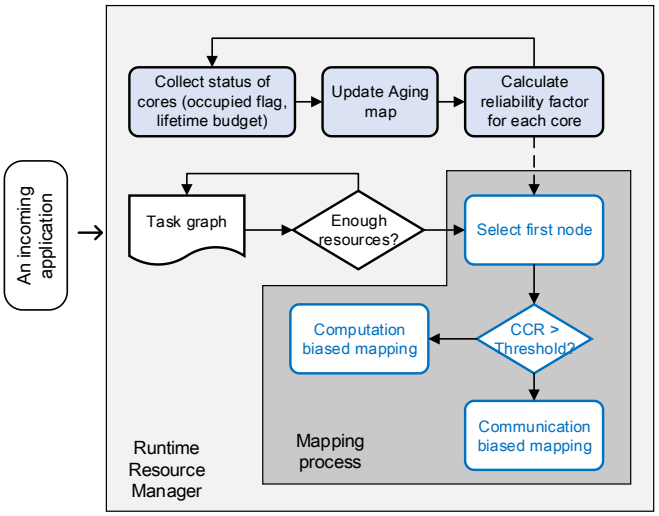


Fig. 3. Flow of the proposed task mapping scheme

where t^i_{arrive} and t^i_{finish} are the arrival time and the finishing time of the i-th application respectively. Waiting time occurs when an application arrives at the system but there are no sufficient resources (cores, power, lifetime, etc.) to run it. The execution time of an application includes the execution and communication of the tasks.

In this paper, the throughput is defined as the average number of applications finished within a time period, at the end of which the lifetime constraint of the system is satisfied.

$$\textbf{Throughput} = \frac{N}{t - t^0_{\text{arrive}}} \tag{6}$$

where t^0_{arrive} is the arrival time of the first application. To improve the throughput, the objective is to

$$\min \quad t \tag{7}$$

subject to

$$t \geq t^{N-1}_{\text{finish}} \tag{8}$$

$$\Delta A_j(t) > 0, \forall n_j \in P \tag{9}$$

Eq. 8 defines the constraint that the all N applications should have finished at time t. Eq. 9 defines the lifetime constraint which should be satisfied at time t, otherwise the high-aged cores should be kept free until the lifetime constraint is met.

V. Proposed Approach for Lifetime Constraint Mapping

In this section, we first propose a lifetime-constraint mapping scheme. Fig. 3 presents the flow for task mapping scheme. First the centralized manager updates the aging map of the many-core system periodically. When a new application arrives, the mapping algorithm is called to map the application onto the many-core system. The mapping algorithm includes two steps (1) first node selection (2) mapping other tasks around the first node. To satisfy the lifetime constraint, the application is put into the waiting queue if there are not enough cores or lifetime budget to map. Then based on the characteristics of the application, the applications are mapped with either computation-biased mapping or communication-biased mapping.

A. Runtime Aging Monitoring and Aging Map Update

For runtime lifetime management, a runtime aging monitoring framework is required to make decisions based on the runtime aging status. In this paper, we adopt a similar method with [26], in which the aging effect is estimated based on aggregate operating conditions, including the temperature, switching activity, voltage, frequency, etc.

Given a target aging effect, the lifetime budget ΔA is calculated for each core. If $\Delta A > 0$, it indicates that the aging effect is less than the target aging effect, i.e. the lifetime constraint of the core is satisfied, and vice versa. The aging map is updated periodically.

B. First Node Selection

Previous studies [27], [12] have developed various methods to select the first node to minimize external congestion and aging effect of many-core system. In this paper, we adopt a similar method with [12]. For each core (m, n) in the many-core system, a reliability factor $RF_{m,n}$ is evaluated based on the status of the neighborhood cores. The reliability factor is represented as follows,

$$RF_{m,n} = \sum_{i=m-r}^{m+r} \sum_{j=n-r}^{n+r} W_{i,j} \times \Delta A_{i,j} \times (r - d + 1) \tag{10}$$

where $W_{i,j} = 0$ if (i, j) is occupied and $W_{i,j} = 1$ if core (i, j) is unoccupied. r is the radius of square and d is the distance from the center. $RF_{m,n}$ incorporates lifetime reliability metric, status of the cores and the range of the square into a single value. And the closer core around (m, n) is given higher weight. A higher $RF_{m,n}$ indicates that there are more free cores and more lifetime budgets around the core (m, n). The first node is the core with the highest reliability factor. Note that the reliability factor values of all cores are calculated at runtime proactively. Thus the first node selection does not affect the running time of the mapping algorithm.

Algorithm 1 Mapping or queuing the incoming application

Input AG_i: the arrival application;
$\boldsymbol{AM}$: ΔA of all cores;
j: First node.
Output Choose Queue() or Map().

1: $C \leftarrow$ the number of cores that are free and $\Delta A > 0$
2: $|T| \leftarrow$ the number of tasks in AG_i
3: **if** $C < |T|$ **then**
4: **Queue**(AG_i)
5: **else if** $\Delta A_j < 0$ **then**
6: **Queue**(AG_i)
7: **else**
8: **Map**(AG_i)
9: **end if**

C. Lifetime Constraint Satisfaction

To satisfy the lifetime constraint, the resource manager needs to decide whether mapping the incoming application or putting the application into the waiting queue. If there is not enough lifetime budget, it is possible to violate the lifetime constraint to map the incoming application. Therefore, we propose a method in Algorithm 1. In the algorithm, we first count the number of cores that are not occupied by any applications and the lifetime budget is positive ($\Delta A > 0$) (line 1). If the C is less than the number of tasks of the incoming application, there are not enough lifetime budgets, thus the incoming application is put into the waiting queue until more resources are available (line 3), i.e. when more cores are free or there are enough lifetime budgets. Since the first node incorporates the lifetime metric of itself and neighbor nodes, the selected first node is the core with the least-aged area around it. If the $\Delta A < 0$, the application is also put into the waiting queue (line 5). Although when $\Delta A > 0$ for the first node it is possible that the neighbor nodes around the first node have $\Delta A < 0$, this can be solved in the following mapping algorithm which adopts different strategies according to the characteristics of applications.

D. Neighborhood Nodes Allocation

After the first node is selected, the tasks of the application are allocated to the neighborhood cores around the first node. Because the applications have different characteristics, they have different communication performance requirements. In this paper, the applications are classified into communication intensive applications and computation intensive applications according to their communication to computation ratio (CCR), which is defined as the average data volume to be sent in one application divided by the average computation workload. Therefore, a higher CCR value indicates that the application performance is more dominated by the communication while a lower CCR value indicates the application performance is more dominated by the computation. The method to classify the applications is presented in Section V-E.

The *borrowing strategy* is adopted to meet the lifetime constraint in a long-term scale. Shown in Fig. 4, the mapping algorithm includes two sub-routines: communication-biased mapping and computation-biased mapping. If the incoming application is a communication intensive application, the communication-biased mapping is called to map the application to a near-square area with lifetime constraint locally relaxed. This is because the communication performance requirement in local time window is higher, the performance is the first priority to optimize. On the other hand, the computation intensive applications are mapped by the computation-biased mapping sub-routine to an area with higher dispersion to free the high-aged cores. Therefore, the over-stressed core can recover from high-aged status. Thus, the lifetime constraint is satisfied in a long-term scale.

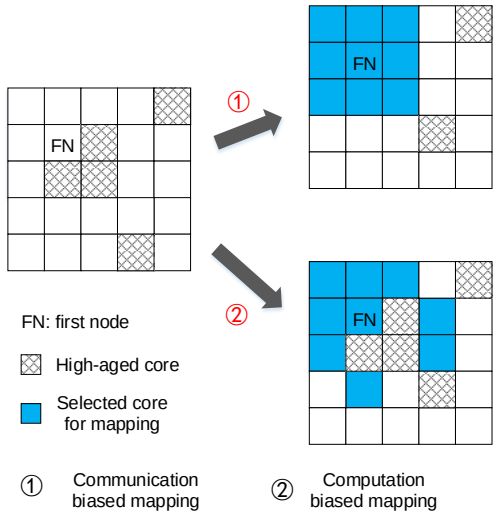


Fig. 4. Diagram for neighborhood nodes allocation, which includes two sub-routines: (1) communication-biased mapping for communication intensive application; (2) computation-biased mapping for computation intensive application.

1) Communication-Biased Mapping: If the application is a communication intensive application, it has higher requirement on the communication performance. Hence the communication distance is more important. The widely adopted method for neighborhood allocation is CONA [27]. With the objective to select a near-square region to mitigate the internal and external congestion [15], CONA selects the node that fits the smallest square when mapping next task. However, the communication distance with predecessors of current task is not considered, which possibly leads to high communication cost.

In this paper, we propose an improved method for neighborhood nodes allocation named lifetime-aware neighborhood allocation (LNA), which takes both task communication overhead and aging information into consideration. The algorithm for the method is presented in Algorithm 2. Given the first node and arrival application, the next task to map is taken from AG_i in breadth first order (line 3 and line 7). The first task is mapped on the first node (line 4). S_r is expanded if there is no available core in current square (line 8-11). If the application is communication intensive, the next core is selected as the core with minimum weighted hops with all predecessors of t_n (line 12-13). Finally, the task t_n is mapped on n_m (line 17). Compared to CONA, the major improvement is that the

Algorithm 2 Pseducode of Lifetime-aware Neighborhood Allocation (LNA)

Input n_f: First node;
$\Delta A_i, i \in [1, |N|]$: Aging map of all cores.
$AG(T, E)$: the application to be mapped;
Output Mapping function $T \rightarrow P$
Definition Ω: the tasks that are not mapped;

1: $\Omega \leftarrow T$ ▷ All tasks in $AG(T, E)$.
2: $r \leftarrow 1$ ▷ Radius of square around the first node
3: Take the first task t_f from Ω in breadth-first order of AG
4: Map t_f on n_f
5: $S_r \leftarrow$ The set of unoccupied cores that are within current square, which is with radius r around n_f
6: **while** $\Omega \neq \varnothing$ **do**
7: Take the next task t_n in Ω in breadth-first order
8: **if** $S_r = \varnothing$ **then**
9: $r \leftarrow r + 1$
10: S_r is updated
11: **end if**
12: **if** communication-biased mapping **then**:
13: $n_m \leftarrow$ the core within S_r and with the minimum number of weighted hops with all predecessors of task t_n
14: **else if** computation-biased mapping **then**:
15: $n_m \leftarrow$ the core with $\Delta A_m > 0$ and within S_r and with the minimum number of weighted hops with all predecessors of task t_n
16: **end if**
17: Map t_n on n_m
18: **end while**

proposed method maps the next task to the core that is with the minimum weighted communication with predecessors of the task. The weighted hops with all predecessors of task t_n can be represented as $\sum w_{i,n} MD(map(t_i), map(t_n)), \forall e_{i,n} \in E$.

Since the maximum possible number of cores in S_r is $|P|$ and the maximum possible number of predecessors is $|T|$, the complexity of the algorithm is $O(|T|^2|P|)$, showing that the algorithm has low complexity.

2) Computation-Biased Mapping: Compared to communication intensive application, the performance of computation intensive application is less affected by the communication distance. This creates the opportunities to map the application to a more dispersed area to avoid high-aged cores. The over-stressed cores can also recover from high-aged status. We propose a method which combines the high-aged cores ($\Delta A < 0$) with other allocated cores to form a near-square shape. The mapping algorithm for computation intensive application is also presented in Algorithm 2 (line 14-16). The high-aged cores are avoided when mapping the application but the communication distance is still considered. Compared to mapping for communication intensive applications, the communication distance increases because more cores are avoided and the mapping area is more dispersed.

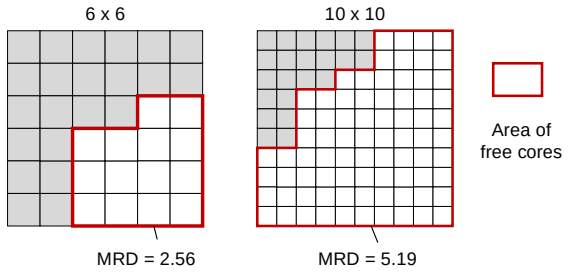
E. Method for Application Classification

As presented in previous section, the runtime mapping scheme includes different strategies for communication intensive applications and computation intensive applications. It still remains unsolved to determine an appropriate threshold to classify the applications. A sub-optimal threshold possibly leads to the following two scenarios:

(1) If the threshold is too low, most applications would be classified as communication intensive applications which are mapped in compact regions, which results from long waiting time as the example shown in Fig. 1;

(2) If the threshold is too high, most applications are classified as computation-intensive applications which are mapped in dispersed regions. The possible scenario is that some applications with high CCR are mapped to dispersed regions, leading to long communication time and high communication cost. The high communication time also results from low throughput.

Above all, an inappropriate threshold can cause low throughput of the system. It is essential to determine an appropriate threshold to effectively improve the throughput of the system. In this paper, we propose a method to determine the threshold dynamically according to communication to computation ratio (CCR) of applications and current locations of all available cores.



(a) 6×6 many-core system (b) 10×10 many-core system

Fig. 5. Average distance of remaining nodes. MRD is the average distance. Grayed blocks represent unavailable cores. The MRD value of the remaining cores increases with the size of many-core.

First, for each application we define CCR as $\frac{comm}{comp}$, which indicates the average communication time in one application divided by the average computation time. A high CCR value means the application is more communication intensive, and a low CCR value means the application is more computation intensive. The next step is how to quantitatively determine a threshold for the CCR to classify the communication intensive applications and computation intensive applications. To avoid the aforementioned 2 scenarios, the threshold should make a balance between the communication time and computation time. Since the threshold should be defined before the mapping process starts, the mapping area and the communication time are unknown. Therefore, it is assumed that all available cores are candidate cores for the mapping when the application is mapped in dispersed region. The average distance of all available cores is calculated and represented as MRD_{free} in Eq. 11

$$MRD = \frac{\sum_{n_i, n_j \in D} MD(n_i, n_j)}{\binom{|D|}{2}} \tag{11}$$

where D denotes the set of all remaining available cores.

TABLE I
CONFIGURATIONS OF SYNTHETIC APPLICATIONS AND PLATFORM

Synthetic graph parameters	
Number of tasks	[10, 20], [20, 30]
Task execution time	[100, 400] (cycles)
Power of task	[50, 100] (mW)
CCR	[0, 0.5], [0, 2]
Network Parameters	
Size & Topology	$8 \times 8, 12 \times 12$ 2D mesh
Routing algorithm	XY routing
Base topology	$8\times$ 8
Latency	2 cycles per hop

$MD(n_i, n_j)$ denotes the Manhattan distance between core n_i and n_j. We define the threshold in Eq. 12.

$$Thsld = \frac{1}{MRD_{free}} \tag{12}$$

Note that $Thsld$ is not a fixed value since it depends on current locations of all available cores. $Thsld$ can approximately make the critical CCR achieve a balance between the communication time and computation time as $MRD_{free} \times comm = comp$. The insight behind this is that the average distance of the tasks is approximately equal to MRD_{free} when the tasks are randomly mapped. $Thsld$ is also associated with topology size. Fig 5 presents a comparison in 6×6 and 8×8 topology respectively. It shows that with the same number of occupied cores, MRD_{free} is much higher in a larger topology. In other words, the computation intensive application is more likely to be mapped to a high dispersed region in a large topology. In this case the threshold should be lower such that more applications are classified as communication intensive applications and mapped to compact regions to avoid high communication distance.

VI. EXPERIMENTAL RESULTS

A. Experimental Setup

In this paper, the experiments are performed in an in-house many-core simulator. McPat [28] and CALIPER [29] are integrated to model the power and lifetime reliability respectively. Both synthetic and realistic task graphs are adopted to evaluate the proposed mapping schemes in the experiments. The synthetic task graphs are generated by DAGGEN [30] with configurations shown in Table I. Other configurations to generate the synthetic graphs are adopted from the default parameters in DAGGEN. Besides we also evaluate the mapping scheme using realistic task graphs which are extracted from video processing applications [31].

The proposed lifetime-constrained mapping schemes *LBRM*, which incorporate CONA and LNA neighborhood allocation methods, are named *LBRM-CONA* and *LBRM-ANA* respectively. The mapping scheme is also compared with the state-of-the-art lifetime-constrained scheme named as *DSRM* [12]. The lifetime-agnostic runtime mapping named *MAPPRO* [24] is also compared.

TABLE II
CONFIGURATIONS FOR EVALUATION

8×8 many-core	A1	A2	A3	A4
Application size	[10, 20]		[30, 40]	
CCR	[0, 2]	[0, 0.5]	[0, 2]	[0, 0.5]

B. Evaluations on Synthetic Applications

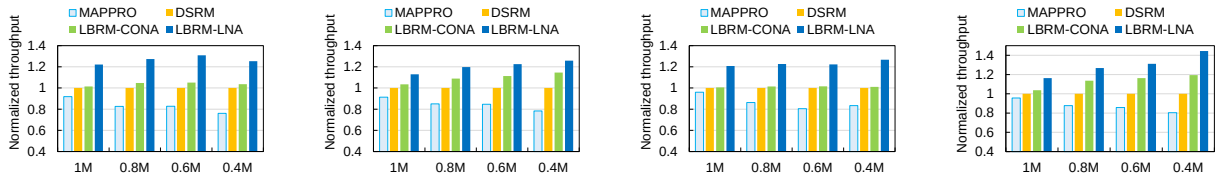
In this subsection, the proposed mapping algorithm is evaluated in terms of throughput and communication cost under various lifetime constraint. There are in total 10^6 applications generated in the simulation, and the arrival rate of application is 0.01 applications per cycle.

1) Throughput Comparisons: We adopt aging effect as the lifetime reliability constraint. Since aging effect reflects the accumulated aging status with the time (shown in Eq. 2), we set the lifetime constraint from $1M$ to $0.4M$ respectively, where M is a reference aging effect value without any lifetime constraints. In other words, the initial lifetime budget is set from $1M$ to $0.4M$, among which $0.4M$ is the tightest constraint imposed on the many-core system.

Fig. 6 and Fig. 7 show the comparisons of normalized throughput in 8×8 and 12×12 many-core system respectively. A1-A4 and B1-B4 correspond to configurations with various application sizes and CCR value as shown in Table II. The throughput is normalized to *DSRM* for comparison. In Fig. 6, when the initial lifetime budget is $1M$, the average throughput improvement of *LBRM* over *DSRM* is 18.1%. The improvement comes from the proposed new neighborhood allocation scheme *LNA*. When the initial lifetime budget is $0.4M$, the average throughput improvement *LBRM-LNA* over *DSRM* reaches 30.1%. This is because of both the adopted *borrowing strategy* and *LNA*. By using *borrowing strategy*, *LBRM* maps computation-intensive applications in dispersed regions without occupying high-aged cores. The throughput is improved due to less waiting time of computation-intensive applications. *LBRM-CONA*, combining *borrowing strategy* with *CONA* shows at most 3.6% and 1.2% throughput improvement for A1 and A3 respectively, and at most 9.5% and 13.2% throughput improvement for A2 and A4 respectively. The reason of lower improvement for A1 and A3 is that when CCR is $[0, 2]$ less applications are classified as computation-intensive applications and the lifetime constraint is over relaxed, making *borrowing strategy* less effective as *LBRM*.

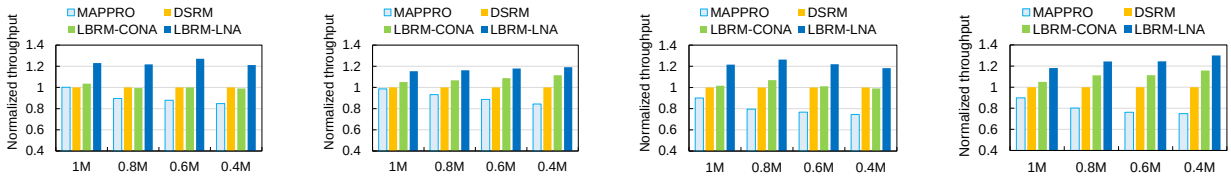
Fig. 7 presents similar results in 12×12 topology size. When the lifetime constraint is $0.4M$, the average throughput improvement *LBRM-LNA* over *DSRM* reaches 22.1%, which is less than that of 8×8 topology size. This is because the relative size of the applications is smaller when in a larger topology, thus creating less opportunities to avoid high-aged cores. Similar as 8×8 topology, *LBRM-CONA* has no significant throughput improvement when the CCR value is $[0, 2]$, but has at most 11.2% and 15.7% throughput improvement for A1 and A4 respectively.

2) Comparisons of Communication Cost: Average Weighted Manhattan Distance (AWMD) is a widely used metric to evaluate the communication cost. Defined in [27],



(a) A1 (b) A2 (c) A3 (d) A4

Fig. 6. Many-core platform size: 8×8. (a)-(d) are the results under various application sizes and CCR values. The results are normalized throughput under various lifetime constraints. $1M$, $0.8M$, $0.6M$, $0.4M$ denote various lifetime constraints. M is reference aging effect value of the system when there is no lifetime constraint. The configurations of A1-A4 is referred in Table II



(a) A1 (b) A2 (c) A3 (d) A4

Fig. 7. Many-core platform size: 12×12. (a)-(d) are the results under various application sizes and CCR values. The results are normalized throughput under various lifetime constraints. $1M$, $0.8M$, $0.6M$, $0.4M$ denote various lifetime constraints. M is reference aging effect value of the system when there is no lifetime constraint. The configurations of B1-B4 is referred in Table II

AWMD is the sum product of communication distance and all edges' weight of the mapped application, averaged by the total communication weights. To evaluation the communication cost of all mapped applications, AWMD defined in this paper considers all mapped applications instead of single application [27]. The AWMD is defined as follows,

$$\text{AWMD} = \frac{\sum_n \sum_{e_{i,j} \in E_n} w_{i,j} \times MD(e_{i,j})}{\sum_n \sum_{e_{i,j} \in E_n} w_{i,j}} \tag{13}$$

where n is the total number of mapped applications, E_n the set of edges of the n-th application, $w_{i,j}$ is the weight of the edge $e_{i,j}$ and $MD(e_{i,j})$ is the Manhattan Distance of task t_i and t_j. A lower AWMD value indicates lower communication cost in terms of communication time and communication energy.

Table III shows the partial comparisons of the AWMD values with respect to Fig. 6 and Fig. 7. Due to space limitations, only AWMD values of $1M$ and $0.6M$ are shown. It shows that in average *LBRM-CONA* leads to slight increase on AWMD compared to *DSRM*. This is because for *LBRM-CONA*, the computation intensive applications are allowed to map to dispersed regions, which increases the communication distance. However, because only low-communication applications are mapped in dispersed regions, the AWMD is less affected. It can also been concluded that *LBRM-CONA* can effectively improve the throughput without significant impact on the communication cost. *LBRM-LNA* has much less AWMD value because an improved neighborhood allocation scheme is adopted, which reduces the communication distance significantly.

C. Throughput Improvement vs. CCR range

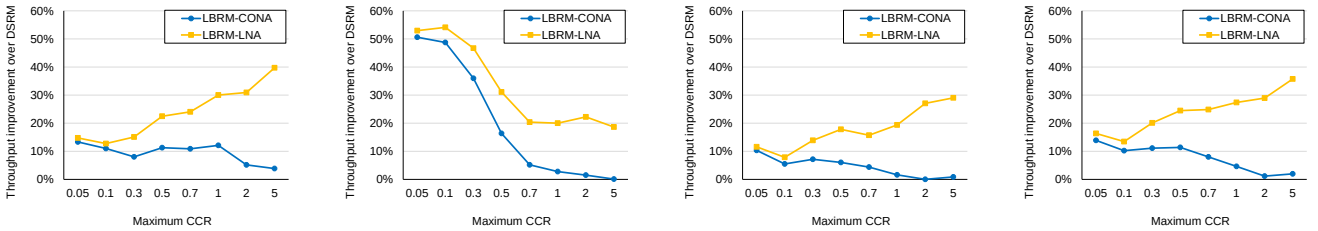
In this section, we characterize the throughput improvement over *DSRM* given different CCR ranges. The CCR range is between 0 and maximum CCR. The maximum CCR is chosen from 0.05 to 5. Fig. 8 presents the results for different topology sizes and applications sizes.

When CCR is low, it can be observed that *LBRM-CONA* can effectively improve the throughput for all configurations. This is because when CCR is low, nearly all applications are classified as computation-intensive applications which can only be mapped to compact area for *DSRM*. However, *LBRM-CONA* relaxes the requirement of compact area and reduces the waiting time of applications. Hence the throughput is improvement. The improvement is even more than 50.6% when the topology size is 8×8 and the application size is $[30, 40]$. The reason is that the application sizes of most applications even exceed half of the topology size, making it more difficult to find a compact area for *DSRM*. The waiting time can be greatly reduced if they are mapped to dispersed regions. When CCR is high, the improvement *LBRM-CONA* is much less obvious because most applications are with high CCR values and most applications are classified as communication intensive applications. *LBRM-CONA* also maps most application to compact area as *DSRM*, thus showing less advantages.

LBRM-CONA improves the throughput mainly due to *borrowing strategy*. Besides *borrowing strategy*, the improvement of *LBRM-LNA* also comes from the improved neighborhood allocation scheme *LNA*. When the CCR is high, it shows that *LNA* brings additional 20%-40% throughput improvement. However, when CCR is low, *LBRM-LNA* is close to *LBRM-CONA*. This is because *LNA* mainly improves the task allocation by considering the communication with all predecessors. If there is less communication, *LNA* is getting closer to *LBRM-CONA*. Fig. 8(b) also shows quite different curve with others because *LBRM* outperforms *DSRM* a lot at low CCR range and relative large application size.

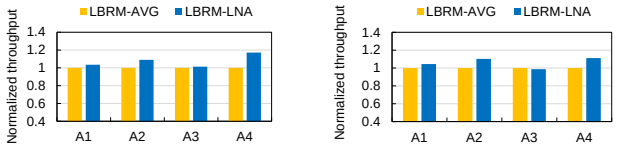
TABLE III
AVERAGE WEIGHTED MANHATTAN DISTANCE

Many-core size	8×8								12×12							
Configurations	A1		A2		A3		A4		A1		A2		A3		A4	
Constraints	1M	0.6M	1M	0.6M	1M	0.6M	1M	0.6M	1M	0.6M	1M	0.6M	1M	0.6M	1M	0.6M
MAPPRO	2.90	2.90	2.93	2.89	4.34	4.33	4.34	4.33	2.95	2.94	2.97	2.92	4.38	4.43	4.41	4.40
DSRM	2.91	2.90	2.88	2.90	4.42	4.42	4.42	4.42	2.97	2.91	2.96	2.90	4.42	4.42	4.45	4.40
LBRM-CONA	2.91	2.91	2.96	3.17	4.42	4.41	4.45	4.35	2.94	2.88	3.05	3.16	4.42	4.29	4.45	4.43
LBRM-LNA	2.02	1.99	2.09	2.28	2.73	2.70	2.74	2.75	2.07	2.17	2.34	2.24	2.89	2.87	2.90	3.02



(a) 8×8, app. size: [10, 20] (b) 8×8, app. size: [30, 40] (c) 12×12, app. size: [10, 20] (d) 12×12, app. size: [30, 40]

Fig. 8. Throughput improvement over *DSRM* vs. CCR range for various topology size and application size. The CCR range is between 0 and maximum CCR. The CCR values of the applications are uniformly distributed within the given range.



(a) Many-core size: 8×8 (b) Many-core size: 12×12

Fig. 9. Comparisons of the Classification Methods

TABLE IV
INFORMATION OF REALISTIC TASK GRAPHS

Name	**Size**	**CCR**	**Name**	**Size**	**CCR**
MP3 Encoder	13	0.008	H.263 Decoder	14	0.008
H.263 Encoder	12	0.093	VOPD	16	1
MWD	12	0.256	MPEG4	12	0.256
PIP	8	0.608	TVOPD	38	0.724

D. Advantages of Proposed Classification Method

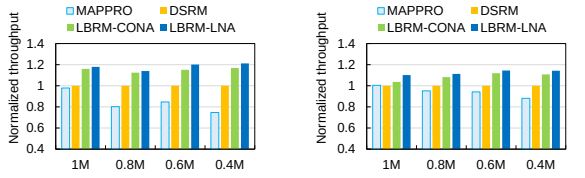
To show the advantages of the proposed classification method, we compare it with a naive method, which determines the threshold according to the average CCR of all applications. For example, the threshold of CCR range $[0, 0.5]$ is 0.25. As shown in Fig. 9, *LBRM* mapping schemes, which incorporate the proposed method and the naive method are named *LBRM-LNA* and *LBRM-AVG* respectively. In Fig. 9, A1-A4 correspond to the configurations in Table II. It shows that when CCR is $[0, 0.5]$, *LBRM-LNA* has more than 10% throughput improvement than *LBRM-AVG*. However, when CCR is $[0, 2]$, the improvement is not significant. This indicate that *LBRM-LNA* outperforms *LBRM-AVG* in most cases. When CCR is $[0, 0.5]$, the threshold determined by the proposed method is more appropriate for high throughput.

E. Evaluations on Realistic Task Graphs

In the paper, the realistic task graphs include Video Object Plane Decoder (VOPD), Picture-In-Picture application (PIP), and Multi Window Display application (MWD), triple video object plane decoder (TVOPD), MPEG4, MP3 encoder, H.263 encoder and H.263 decoder [31], [32], [33]. Since only communication profiles of these applications are provided, we assume that the computation time of the tasks is identical and the CCR value of VOPD is 1. The CCR values of the other applications are normalized accordingly shown in Table IV. In the simulation, there are in total 10^6 applications in which the 8 task graphs are uniformly distributed.

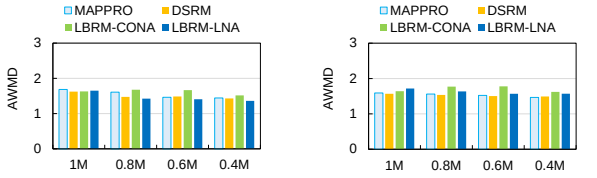
1) Throughput Comparisons: Fig. 10 shows the comparisons of throughput for realistic task graphs. It shows that in 8×8 topology size, *LBRM-CONA* and *LBRM-LNA* has at most 16.8% and 21.2% throughput improvement respectively. When the topology size is 12×12, *LBRM-CONA* and *LBRM-LNA* has at most 11.9% and 14.4% throughput improvement respectively. *LBRM-LNA* has higher throughput because *borrowing strategy* is also adopted besides LNA neighborhood allocation. The improvement is less in 12×12 because the relative size of an application is smaller compared to the platform, creating less chances for avoiding high-aged cores. The results also show that *LBRM* is also effective in terms of throughput improvement for realistic task graphs.

2) AWMD Comparisons: Fig. 11 shows the comparisons of communication distance for realistic task graphs. The results show that although *LBRM-CONA* and *LBRM-LNA* allow dispersed mapping of some applications, the AWMD does not increase significantly. The main reason is that most of the dispersed applications have low communication. We can conclude that *LBRM* can effectively increase the throughput with only a little penalty on the communication cost.



(a) Many-core size: 8×8 (b) Many-core size: 12×12

Fig. 10. Throughput comparison for realistic benchmarks



(a) Many-Core size: 8×8 (b) Many-Core size: 12×12

Fig. 11. AWMD Comparisons for realistic benchmarks

VII. Conclusions

In this paper, we propose a lifetime-constrained runtime mapping algorithm, which can dynamically map the incoming applications on the many-core system under a lifetime constraint. Compared to existing lifetime aware constrained mapping, the proposed mapping algorithms adopt the *borrowing strategy* to manage the many-core resources in multi-scale. In other words, the lifetime is managed in long-term scale but is locally relaxed to satisfy performance requirement in short-term scale. The applications are classified into communication intensive and computation intensive applications, and mapped with different strategies. We also propose an effective method to classify the application into communication intensive and computation intensive applications. The experiments show that compared to state-of-the-art lifetime-constrained mapping scheme, the proposed scheme can effectively improve the throughput on many-core system for synthetic task graphs and realistic task graphs while having less impact on the communication cost.

References

[1] C. Ramey, "Tile-gx100 manycore processor: Acceleration interfaces and architecture," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, Aug 2011, pp. 1–21.

[2] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, "Knights landing: Second-generation intel xeon phi product," *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.

[3] W. Song, S. Mukhopadhyay, and S. Yalamanchili, "Architectural reliability: Lifetime reliability characterization and management of many-core processors," *IEEE Computer Architecture Letters*, vol. 14, no. 2, pp. 103–106, 2015.

[4] L. Huang and Q. Xu, "Characterizing the lifetime reliability of manycore processors with core-level redundancy," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '10. Piscataway, NJ, USA: IEEE Press, 2010, pp. 680–685. [Online]. Available: http://dl.acm.org/citation.cfm?id=2133429.2133574

[5] A. M. Rahmani, M. H. Haghbayan, A. Miele, P. Liljeberg, A. Jantsch, and H. Tenhunen, "Reliability-aware runtime power management for many-core systems in the dark silicon era," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 2, pp. 427–440, 2017.

[6] Z. Yang, C. Serafy, T. Lu, and A. Srivastava, "Phase-driven learning-based dynamic reliability management for multi-core processors," in *Proceedings of the 54th Annual Design Automation Conference (DAC)*, 2017, pp. 46:1–46:6.

[7] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, "The case for lifetime reliability-aware microprocessors," in *Proceedings of the 31st Annual International Symposium on Computer Architecture (ISCA)*, 2004, pp. 276–285.

[8] D. Lorenz, G. Georgakos, and U. Schlichtmann, "Aging analysis of circuit timing considering nbti and hci," in *Proceedings of 15th IEEE International On-Line Testing Symposium*, 2009, pp. 3–8.

[9] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and user experience aware dynamic reliability management in multicore processors," in *Proceedings of 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[10] L. Huang, F. Yuan, and Q. Xu, "Lifetime reliability-aware task allocation and scheduling for mpsoc platforms," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2009, pp. 51–56.

[11] C. Bolchini, M. Carminati, A. Miele, A. Das, A. Kumar, and B. Veeravalli, "Run-time mapping for reliable many-cores based on energy/performance trade-offs," in *Proceedings of International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, 2013, pp. 58–64.

[12] M. H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2016, pp. 854–857.

[13] L. Huang and Q. Xu, "Energy-efficient task allocation and scheduling for multi-mode mpsocs under lifetime reliability constraint," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010, pp. 1584–1589.

[14] D. Gnad, M. Shafique, F. Kriebel, S. Rehman, D. Sun, and J. Henkel, "Hayat: Harnessing dark silicon and variability for aging deceleration and balancing," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 180:1–180:6.

[15] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *Proceedings of 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2013, pp. 1–6.

[16] C. Zhuo, D. Sylvester, and D. Blaauw, "Process variation and temperature-aware reliability management," in *Proceedings of Design, Automation and Test in Europe Conference Exhibition*, 2010, pp. 580–585.

[17] E. Karl, D. Blaauw, D. Sylvester, and T. Mudge, "Reliability modeling and management in dynamic microprocessor-based systems," pp. 1057–1060, 2006.

[18] L. Wang, X. Wang, H.-f. Leung, and T. Mak, "Throughput optimization for lifetime budgeting in many-core systems," in *Proceedings of the on Great Lakes Symposium on VLSI*, 2017, pp. 451–454.

[19] B. H. Meyer, A. S. Hartman, and D. E. Thomas, "Cost-effective slack allocation for lifetime improvement in noc-based mpsocs," in *Proceedings of Design, Automation Test in Europe Conference Exhibition (DATE)*, 2010, pp. 1596–1601.

[20] C. Zhu, Z. Gu, R. P. Dick, and L. Shang, "Reliable multiprocessor system-on-chip synthesis," in *Proceedings of 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 239–244.

[21] A. S. Hartman and D. E. Thomas, "Lifetime improvement through runtime wear-based task mapping," in *Proceedings of the 10th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2012, pp. 13–22.

[22] V. Rathore, V. Chaturvedi, and T. Srikanthan, "Performance constraint-aware task mapping to optimize lifetime reliability of manycore systems," in *Proceedings of the 26th Great Lakes Symposium on VLSI(GLVLSI)*, 2016, pp. 377–380.

[23] J. Sun, R. Lysecky, K. Shankar, A. Kodi, A. Louri, and J. Roveda, "Workload assignment considering nbti degradation in multicore systems," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 10, no. 1, pp. 4:1–4:22, 2014.

[24] 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 *Proceedings of the 9th International Symposium on Networks-on-Chip (NoCS)*, 2015, pp. 26:1–26:8.

[25] M. Fattah, P. Liljeberg, J. Plosila, and H. Tenhunen, "Adjustable contiguity of run-time task allocation in networked many-core systems," in

2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC), 2014, pp. 349–354.

[26] P. Mercati, A. Bartolini, F. Paterna, L. Benini, and T. S. Rosing, "An online reliability emulation framework," in *Proceedings of the 12th IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, 2014, pp. 334–339.

[27] M. Fattah, M. Ramirez, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Cona: Dynamic application mapping for congestion reduction in many-core systems," in *Proceedings of 30th International Conference on Computer Design (ICCD)*, 2012, pp. 364–370.

[28] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[29] C. Bolchini and et al., "A lightweight and open-source framework for the lifetime estimation of multicore systems," in *Proceedings of International Conference on Computer Design (ICCD)*, 2014, pp. 166–172.

[30] F. Suter, "Daggen: A synthethic task graph generator," https://github.com/frs69wq/daggen.

[31] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. D. Micheli, "Noc synthesis flow for customized domain specific multiprocessor systems-on-chip," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 16, no. 2, pp. 113–129, 2005.

[32] Y. Xue and P. Bogdan, "Improving noc performance under spatio-temporal variability by runtime reconfiguration: a general mathematical framework," in *Proceedings of Tenth IEEE/ACM International Symposium on Networks-on-Chip (NOCS)*, 2016, pp. 1–8.

[33] S. Murali, C. Seiculescu, L. Benini, and G. D. Micheli, "Synthesis of networks on chips for 3d systems on chips," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2009, pp. 242–247.

PLACE PHOTO HERE

Liang Wang

PLACE PHOTO HERE

Xiaohang Wang

PLACE PHOTO HERE

Ho-fung Leung

PLACE PHOTO HERE

Terrence Mak

PLACE PHOTO HERE

Jie Han

PLACE PHOTO HERE

Leibo Liu