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 have can have up to 50.6% throughput improvement with only a small penalty on the communication cost.

Index Terms—IEEE, IEEEtran, journal, L\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 different 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.

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 (TDBB), 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 $3 \times 3$ many-core system. 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, which possibly lead to low

1It 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.
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 $\alpha$ 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

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

where $\alpha$ is related to operating conditions such as temperature, switching activity, frequency, etc. $\Gamma$ is the gamma function. $\beta$ 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 $\Delta t_i$ is the duration of the $i$-th time interval. $\alpha_i$ 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.

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

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. $A_{\text{target}}(t)$ 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 $\Delta A$ value. The resource management manager maintains an aging map, which contains the $\Delta A$ 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.

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

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 $\text{map}(t_i)$ 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 $\Delta A$ 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

$$\sigma_i = t_{i,\text{finish}} - t_{i,\text{arrive}} \tag{5}$$
where \( t^i_{\text{arrive}} \) and \( t^i_{\text{finish}} \) 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.

\[
\text{Throughput} = \frac{N}{t - t_{\text{arrive}}^0} \tag{6}
\]

where \( t_{\text{arrive}}^0 \) is the arrival time of the first application. To improve the throughput, the objective is to

\[
\min t \tag{7}
\]

subject to

\[
t \geq t_{\text{finish}}^N \tag{8}
\]

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

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 \( \Delta A \) 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.
High applications and the lifetime budget is positive (first count the number of cores that are not occupied by any we propose a method in Algorithm 1. In the algorithm, we needs to decide whether mapping the incoming application. Therefore, there is not enough lifetime budget, it is possible to violate the or putting the application into the waiting queue. If there 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.

Algorithm 1 Mapping or queuing the incoming application

\[
\text{Input} \quad AG_i: \text{the arrival application};
\]

\[
\Delta M: \Delta A \text{ of all cores};
\]

\[
j: \text{First node}.
\]

\[
\text{Output} \quad \text{Choose } \text{Queue()} \text{ or } \text{Map()}.\]

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: \( \text{Queue}(AG_i) \)
5: else if \( \Delta A_j < 0 \) then
6: \( \text{Queue}(AG_i) \)
7: else
8: \( \text{Map}(AG_i) \)
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.

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 computation 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 Pseudocode 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 \( \Omega \): the tasks that are not mapped;

1: \( \Omega \leftarrow T \) \hspace{1cm} \triangleright \text{All tasks in } AG(T, E).
2: \( r \leftarrow 1 \) \hspace{1cm} \triangleright \text{Radius of square around the first node}
3: Take the first task \( t_f \) from \( \Omega \) 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: \textbf{while} \( \Omega \neq \emptyset \) \textbf{do}
7: \hspace{1cm} Take the next task \( t_n \) in \( \Omega \) in breadth-first order
8: \hspace{1cm} \textbf{if} \( S_r = \emptyset \) \textbf{then}
9: \hspace{2cm} \( r \leftarrow r + 1 \)
10: \hspace{1cm} \text{\( S_r \) is updated}
11: \hspace{1cm} \textbf{end if}
12: \hspace{1cm} \textbf{if} communication-biased mapping \textbf{then}
13: \hspace{1.5cm} \( n_m \leftarrow \) the core within \( S_r \) with the minimum number of weighted hops with all predecessors of task \( t_n \)
14: \hspace{1cm} \textbf{else if} computation-biased mapping \textbf{then}
15: \hspace{1.5cm} \( n_m \leftarrow \) the core with \( \Delta A_m > 0 \) and within \( S_r \)
\hspace{1.5cm} \text{and with the minimum number of weighted hops with all}
\hspace{1.5cm} \text{predcessors of task } t_n
16: \hspace{1cm} \textbf{end if}
17: \hspace{1cm} Map \( t_n \) on \( n_m \)
18: \textbf{end while}

The 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 \( \sum \omega_{i,n} MD(map(t_i), map(t_n)), \forall e_{i,n} \in E \).

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 overstressed 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.

First, for each application we define CCR as \( \frac{\text{comm}}{\text{comp}} \), 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 \( \text{MRD}_{\text{free}} \) in Eq. 11

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

where \( D \) denotes the set of all remaining available cores.
$MD(n_i, n_j)$ denotes the Manhattan distance between core $n_i$ and $n_j$. We define the threshold in Eq. 12.

$$Thsld = \frac{1}{MRD_{\text{free}}}$$ (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 $MRD_{\text{free}} \times \text{comm} = \text{comp}$. The insight behind this is that the average distance of the tasks is approximately equal to $MRD_{\text{free}}$ when the tasks are randomly mapped. $Thsld$ is also associated with topology size. Fig 5 presents a comparison in $6 \times 6$ and $8 \times 8$ topology respectively. It shows that with the same number of occupied cores, $MRD_{\text{free}}$ 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-LNA 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>
<thead>
<tr>
<th>Synthetic graph parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of tasks</td>
</tr>
<tr>
<td>Task execution time</td>
</tr>
<tr>
<td>Power of task</td>
</tr>
<tr>
<td>CCR</td>
</tr>
<tr>
<td>Network Parameters</td>
</tr>
<tr>
<td>Size &amp; Topology</td>
</tr>
<tr>
<td>Routing algorithm</td>
</tr>
<tr>
<td>Base topology</td>
</tr>
<tr>
<td>Latency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application size</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
</tr>
<tr>
<td>Application size</td>
</tr>
<tr>
<td>CCR</td>
</tr>
</tbody>
</table>

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 $8 \times 8$ and $12 \times 12$ 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 $12 \times 12$ 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 $8 \times 8$ 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 $8 \times 8$ 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],
AWMD is the sum product of communication distance and all edges’ weight of the mapped application, averaged by the total communication weights. To evaluate 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,

$$\text{AWMD} = \frac{\sum_n \sum_{e_{i,j} \in E_n} w_{i,j} \times MD(e_{i,j})}{\sum_n \sum_{e_{i,j} \in E_n} w_{i,j}}$$  \hspace{1cm} (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 be 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 \(8 \times \) 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

<table>
<thead>
<tr>
<th>Many-core size</th>
<th>8 × 8</th>
<th>12 × 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configurations</td>
<td>A1</td>
<td>A2</td>
</tr>
<tr>
<td>Constraints</td>
<td>1M</td>
<td>0.6M</td>
</tr>
<tr>
<td>MAPPRO</td>
<td>2.90</td>
<td>2.90</td>
</tr>
<tr>
<td>DSRM</td>
<td>2.91</td>
<td>2.88</td>
</tr>
<tr>
<td>LBRM-CONA</td>
<td>2.91</td>
<td>2.91</td>
</tr>
<tr>
<td>LBRM-LNA</td>
<td>2.02</td>
<td>1.99</td>
</tr>
</tbody>
</table>

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) 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. 9. Comparisons of the Classification Methods

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 10³ 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.
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.

VII. CONCLUSIONS

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

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

REFERENCES


