2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems

Topology Mapping of Parallel Applications onto Random Allocations

Yao Hu
National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430
huyao0107@gmail.com

***Abstract*—In high-performance computing (HPC) systems, one application usually has many parallel tasks running on multiple compute nodes. The execution time depends on the communication latency and the network contention between the parallel tasks especially when these tasks intensively communicate with each other. Traditionally, the tasks are mapped onto a regular topology with nearby compute nodes to reduce the network distances. In this case, fragmentation of unused compute nodes cannot be assigned for a newly incoming job, because it is assumed to largely harm the communication abilities between non-adjacent nodes. However, the communication overhead between communicating tasks is also determined by the application's communication pattern, which may not suit to a specified regular topology. For the purpose of improving job scheduling abilities, we investigate the job mapping on random topologies so that a newly incoming job can be immediately dispatched as long as there are enough available nodes. Evaluation results show that, for a large compound workload of NAS Parallel Benchmarks (NPB) applications, the random job mapping can reduce up to 64% of makespan and up to 80% of turnaround time when compared with the regular job mapping. Overall, the random topology embedding in random topologies indicates substantial room for improvement of job scheduling performance.**

***Index Terms*—High-performance computing (HPC), topology embedding, interconnection network, job scheduling**

I. Introduction

In high-performance computing (HPC) systems, a number of highly parallel applications simultaneously run on multiple compute nodes [1]. The both scales of compute nodes and applications have been growing exponentially in the last decade [2]. Mapping virtual inter-*process* topologies to physical inter-*processor* topologies is one of the most important issues on datacenters and supercomputers. Traditionally, the compute nodes form a fixed *host* network topology, and the user jobs are allocated to unused compute node set connected by regular *guest* network topologies. For instance, the custom supercomputer Cray XT6 [3] is connected by a 3-D torus network topology and Blue Gene/Q [4] is connected by a 5-D torus network topology. On the both supercomputer systems, each job is executed on a (small) embedded k-ary n-cube interconnection network. By dispatching communicating tasks to nearby compute nodes, the communication latency and network contention can be minimized. Such traditional regular topology mapping can benefit from low-latency high-bandwidth communication between nearby compute nodes [5].

One disadvantage of the traditional topology mapping on regular topologies is that, when the instantaneous workload becomes large the system utilization of compute nodes would decrease. In this case, a number of jobs have to wait for the release of occupied adjacent compute nodes, because they cannot be mapped on non-adjacent compute nodes even when they are available. In other words, there are many cases where the system utilization is low while some available compute nodes are not allowable for user jobs since they are disjoint [6]. It is a time-space tradeoff that the regular topology mapping tends to minimize the execution time but wasting the space of non-adjacent compute nodes.

On the other hand, the topology mapping on regular topologies is not always the best choice for every application especially when its communication pattern does not suit the regular job mapping. For example, the application of Matrix Multiplication (MM) is executed longer on regular mesh and torus topologies than on random topologies (see Section IV-A). In many current open-source resource and job management systems (RJMSs) or batch schedulers [7], network topology characteristics are taken into account to favor the choice of compute nodes that are placed close to each other so as to avoid long-distance communications [8]. However, they eventually fail to take into consideration the application behavior when allocating compute nodes.

In this study, we explore a simple but efficient topology mapping strategy which is designed to use the maximum number of compute nodes on the system. We investigate the time-space tradeoff by mapping each application on a random topology. In our previous study [9], we have analyzed the impact of topology mapping *dilation* on job scheduling performance. However, we assumed that the link bandwidth is unbounded, i.e., we did not consider the network *contention* between parallel jobs. In this work, in addition to the mapping *dilation*, we also take into consideration the network *contention* to evaluate its impact on the job scheduling performance. The problem of job mapping on random topologies for HPC systems is very complicated due to connection irregularity and routing complexity [10] [11]. In this context, the job mapping on a random topology is considered only when a regular *guest* topology of unused compute nodes cannot be found. Therefore, the fragmentation of unused compute nodes caused by previous job allocation can be used for constructing a random topology for next incoming job so that they can be simultaneously executed on the system. Ideally, a workload of diverse applications can be better supported disregarding its interconnection network with a certain time-space tradeoff.

Figure 1 depicts an example of job mapping on a regular 2-D mesh network and a random topology. For the former regular mapping (left in Fig. 1), one job cannot be dispatched to the available compute nodes that cannot form a 2-D mesh network. For the latter random mapping (right in Fig. 1), one job can be still mapped on the available compute nodes even though they do not form a regular network, e.g., 2-D mesh. It seems that the job mapping on random topologies can simultaneously accommodate a larger number of user jobs, but it may impose additional execution time on dispatched jobs. Our target in this work is to investigate the job scheduling performance by mapping jobs on random topologies instead of traditional regular topologies.

978-1-7281-2058-4/19/$31.00 ©2019 IEEE
DOI 10.1109/HPCC/SmartCity/DSS.2019.00199

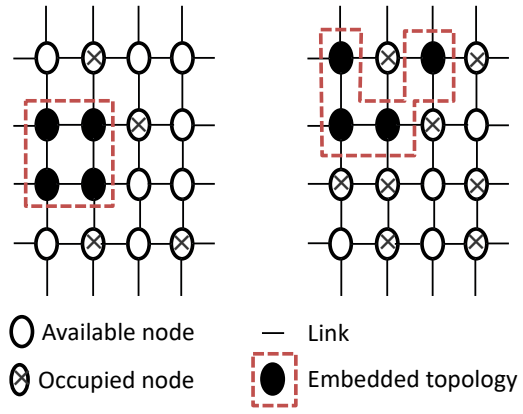


Figure 1. The job mapping on a regular (2-D mesh) network (left) and a random topology (right).

Our main contributions in this work are:

- We investigated the impact of job mapping on random topologies where even *non-contiguous* compute nodes can be used to construct physical inter-processor topologies (Section IV-A and IV-B).
- We evaluated the impact of network *contention* between simultaneously running applications on their execution times (Section IV-C).
- By an event-driven simulation, we evaluated the job scheduling performance of random topology embedding. Evaluation results showed that the random job mapping can reduce up to 64% of makespan and up to 80% of turnaround time when compared with the regular job mapping for a compound workload of NPB applications (Section IV-D).

The rest of this paper is organized as follows. Background information and related works are discussed in Section II. Section III analyzes the job mapping on regular interconnection networks and random topologies. Section IV presents our evaluation including simulation settings and experiment results. Section V concludes this study with a summary of our findings in this paper.

II. Background and Related Works

A. The Mapping Problem

Job mapping and its family can be equivalent to topology embedding in the field of graph theory, thus we use a notation that extends that used for graph embedding [12].

We represent the logical inter-*process* communication pattern using a graph $C = (V_c, E_c)$, where V_c is the set of processes, and $E_c(u, v)$ is the volume of communication from process $u \in V_c$ to process $v \in V_c$. The value of $E_c(u, v)$ is zero if no communication occurs between process u and process v. The graph C might be disconnected, and isolated vertices can exist, which represent the concurrent execution of multiple unrelated jobs.

Likewise, the physical inter-*processor* interconnection network is represented by a graph $G = (V_g, E_g)$, where V_g is the set of physical compute nodes, and $E_g(m, n)$ is the bandwidth capacity of the link connecting node $m \in V_g$ to node $n \in V_g$. The value of $E_g(m, n)$ is zero if there is no link between node m and node n. Let $P(m, n)$ be the set of simple paths connecting node m to node n, where each edge occurs at most once. Let $T(m, n, p)$ be the fraction of traffic from node m to node n, which is routed through path $p \in P(m, n)$. The traffic is often routed via the shortest path.

A topology mapping is specified by a function F: $V_c \rightarrow V_g$, which maps the vertices (processes) of C to the vertices (processors or nodes) of G. In this context, we define two metrics for a mapping: *dilation* and *contention*. Let $|p|$ denote the length of path p. The *dilation* of an edge (u, v) of the communication graph C is defined as follows:

$$dilation(u, v) = \sum_{p \in P(F(u), F(v))} T(F(u), F(v), p) \cdot |p| \quad (1)$$

It is the average length of paths taken by a message sent from process u to process v. We define the largest *dilation* of all the inter-*process* communications as the *dilation* of the mapping F:

$$dilation(F) = \max_{u,v \in V_c} dilation(u, v) \quad (2)$$

It is the maximum number of edges traversed by packets, and thus has a significant impact on the application execution time. It is seen as a measure of the topology embedding over the interconnection network.

The *contention* of a link (m, n) of the interconnection network is the ratio of the amount of traffic on that link to the capacity of the link. Usually for simplicity, the link capacity is set to one. Hence, the *contention* of an edge $e \in E_g$ is defined as follows:

$$contention(e) = \sum_{u,v \in V_c} \sum_{p \in P(F(u), F(v)), e \in p} T(F(u), F(v), p) \quad (3)$$

Similarly, we define the largest *contention* of all the inter-*processor* communications as the *contention* of the mapping F:

$$contention(F) = \max_{e \in E_g} contention(e) \quad (4)$$

It has a significant impact on the lower bound of the communication latency between communicating tasks.

The both *dilation* and *contention* can be computed in polynomial time. A simple job mapping requires the *dilation* and *contention* of the topology embedding to be one. The conventional regular job mapping is usually such a case. However, for job mapping on random topologies, such requirements are difficult to be satisfied. Especially when one or more than one link is shared by multiple jobs, the *dilation* and *contention* are both larger than one [2]. In this work, we take this case into account when mapping jobs on random topologies. Notice that, in this context a random topology can be *contiguous* or *non-contiguous*, which will be elaborated in Section III. We use the terms mapping and embedding interchangeably.

B. Job Mapping for HPC Applications

In many recent supercomputers, the mapping problem is constrained to mapping the mesh/torus over another mesh/torus [13]. However they have shown that the system utilization is usually low due to node fragmentation. Several general-purpose job mapping schemes have been proposed over the years. For example, Vogelstein et al. proposed the

fast approximate quadratic (FAQ) assignment algorithm [14], which is too expensive for many large-scale HPC problems in terms of computation cost. Agarwal et al. proposed an algorithm [15] designed for mapping and load balancing of parallel applications, which has a similar complexity compared to FAQ. Hoefler et al. proposed three general algorithms [16] intended for mapping of large-scale applications. They evaluated the algorithms using a congestion metric and sparse-matrix vector multiplication code, but did not evaluate the mapping using production HPC applications.

There are also many existing job mapping schemes which are designed for specific applications and networks. For instance, the TARA [17] used a description of application to allocate resources, which is tailored for a very specific class of applications. Jingjin Wu et al. [18] introduced a hierarchical task mapping strategy for modern supercomputers based on generic recursive algorithms for both fat-tree and torus network topologies, which shows good performance with low overhead. Rashti et al. [19] proposed a weighted graph model for the whole physical topology of the computing system, including both the inter and intra node topologies. They showed interesting results with application sets, which are not integrated with real RJMSs or tested with real workload traces. Albing. Carl et al. [20] presented a study for torus network topology, which shows how processor ordering takes place based on space filling curve to map the nodes of the torus onto a 1-dimensional list in order to preserve locality information. Yang et al. [21] presented a window-based locality-aware job scheduling strategy only for torus topology, which tries to optimize the job and system performance at the same time. Its goal is to preserve node contiguity by considering multiple jobs for scheduling while making use of the 0-1 Multiple Knapsack problem for resource allocation.

In our case, we simply investigate only the interconnection network for both the allocation and the execution of user jobs with diverse application behaviors. We do not use any latency or bandwidth figures to compute job allocation.

III. Job Mapping on Random Topologies

In this section, we describe the case that a job is mapped on a random topology when no enough available compute nodes form a regular topology embedding. For ease of understanding, we divide the case into several scenarios according to different *host* networks and *guest* topologies. The *host* network is defined as the entire inter-*process* interconnection network of the target system, and a *guest* topology is defined as an inter-*processor* topology assigned to an incoming user job. Notice that, in this study a regular topology refers to a *contiguous* topology, while a random topology can be *contiguous* or *non-contiguous*.

A. Host Topology: Torus

First, we assume a practical case that the *host* interconnection network is 3-D torus for supercomputers such as Cray XT6 [3]. Remember that the job mapping on random topologies is considered only when current regular mapping does not exist.

We investigate a random topology embedding with dilation-d (d >1). Remember that the *dilation* in the topology embedding refers to the length (in number of hops) of the shortest path between two embedded vertices of an edge. Basically, if the specified dilation d increases, its job mapping becomes easier while imposing larger communication latency between non-adjacent compute nodes. As shown in Fig. 2, a dilation-1 topology embedding is first considered for job mapping if it exists on the current system. This step is equivalent to a regular job mapping. If such a dilation-1 topology embedding cannot be found on the current system, we consider a dilation-2 topology embedding for job execution. If a dilation-2 topology embedding is still not found, we consider a dilation-d (d >2) topology embedding for job execution. Considering that arbitrary node separation during job mapping may harm job scheduling performance due to high communication latency between non-adjacent compute nodes, we map jobs on neighboring compute nodes as many as possible. In this example, only node *C* is non-adjacent to other three nodes.

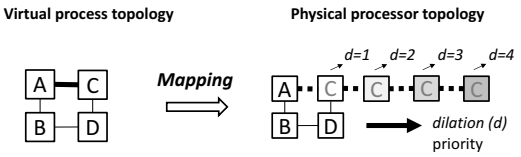


Figure 2. The job mapping with different embedding dilations.

Algorithm 1 Job mapping on random topologies over torus

```
1: procedure MAP_RANDOM_TORUS(Network nw, Job job)
2:     dilation ← 1
3:     if map(nw, dilation) == true then
4:         dispatch(job)
5:     else
6:         while dilation ≤ diameter(nw) - 1 do
7:             dilation ← dilation + 1
8:             if map(nw, dilation) == true then
9:                 dispatch(job)
10:                break
11:            end if
12:        end while
13:    end if
14: end procedure
```

Note that, a dilation-d ($d \leq 2$) topology embedding can be a *contiguous* mapping, while a dilation-d (d >2) topology embedding is a *non-contiguous* mapping [2], where one or more links are shared by multiple jobs as part of compute nodes are not directly connected in the topology embedding. Algorithm 1 describes the entire algorithm to map jobs on random topologies over torus.

B. Host Topology: WS Small-world Network

We assume that the *host* interconnection network is a connected Wattsa-Strogatz (WS) small-world network [22], where the edges are established between adjacent vertices while part of them are randomly defined. One characteristic of the WS small-world network is that the average distance between any two vertices, in large networks, does not exceed a small number of hops [23] [24]. Such connections guarantee few separation hops among the network vertices in large scale.

A WS small-world network is created as follows. First we create a ring over n nodes. Then each node in the ring is joined to its k nearest neighbors (or $k-1$ neighbors if k is odd). Then shortcuts are created by replacing some edges as follows: for each edge (u, v) in the underlying n-ring with k

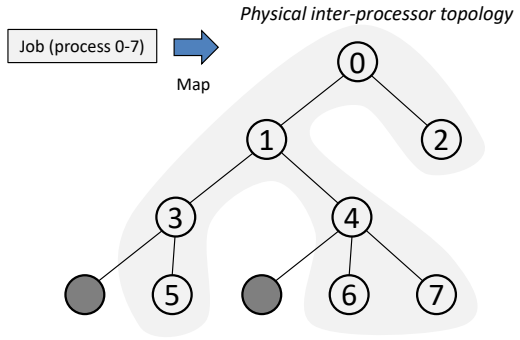


Figure 3. The job mapping on a regular contiguous topology.

nearest neighbors with probability p replace it with a new edge (u, w) with uniformly random choice of existing node w. The random rewiring does not increase the number of edges.

We investigate two patterns of the topology embedding:

1) Regular: A regular topology embedding can be obtained as follows. Assume that an incoming job requires c compute nodes for its execution, which is less than the number of nodes over the *host* topology. If one compute node is not occupied by any job, it is available; otherwise, it is not available. All compute nodes are categorized into two groups: *a_nodes* and *n_nodes*. At the initial stage, the node set *a_nodes* is empty and the node set *n_nodes* contains all the nodes over the *host* topology. An ergodic query is performed within *n_nodes* to search an available node, which will be removed from *n_nodes* and added to *a_nodes*. Then the first node element of *a_nodes* is picked, and an ergodic query is performed within its adjacent nodes to search available nodes. If such an available node is inside *n_nodes*, it will be removed from *n_nodes* and added to *a_nodes*. If the number of nodes in *a_nodes* is less than c, the next (second) node element of *a_nodes* is picked, and an ergodic query is performed again within its adjacent nodes to search available nodes. Similarly, the available nodes are moved from *n_nodes* to *a_nodes*. The algorithm performs such a recursive categorization and stops when the number of nodes in *a_nodes* is equal to c. In this way, the resulting *contiguous* topology embedding is a tree-like *guest* topology, as shown in Fig. 3. Algorithm 2 describes the entire algorithm to map jobs on regular topologies over WS small-world networks.

2) Random: A random topology embedding can be easily obtained by a simple ergodic query on all the compute nodes over the *host* topology. That is, any compute node can be used for such an ideal job mapping, so that a required subgraph can be easily found to accommodate each incoming job as long as there are enough available compute nodes on the current system. This method reduces the wait time for each job in the queue but may increase the execution time due to a large-dilation topology embedding. Notice that, in a *non-contiguous* random embedding as shown in Fig. 4, link sharing may or may not occur against another simultaneously running job. In the former case (*contention* >1), the network bandwidth *contention* may hamper the execution times of the involved parallel jobs; in the latter case (*contention* = 1), such things will not happen. We will show how the link sharing influence the simultaneously running jobs in a *non-contiguous* random embedding in Section IV-B.

Algorithm 2 Job mapping on regular topologies over WS small-world networks

```
1:  procedure MAP_REGULAR_WS(Network nw, Job job)
2:      c ← num_nodes(job)
3:      a_nodes ← {}
4:      n_nodes ← node_set(nw)
5:      index ← 0
6:      for node in n_nodes do
7:          if node is available then
8:              a_nodes.add(node)
9:              n_nodes.remove(node)
10:             break
11:         end if
12:     end for
13:     while num(a_nodes) < c do
14:         current_node ← a_nodes[index]
15:         adjacent_nodes ← current_node.neighbors()
16:         for node in adjacent_nodes do
17:             if node is available in n_nodes then
18:                 a_nodes.add(node)
19:                 n_nodes.remove(node)
20:                 if num(a_nodes) == c then
21:                     break
22:                 end if
23:             end if
24:         end for
25:         index ← index + 1
26:     end while
27:     dispatch(job)
28: end procedure
```

C. Host Topology: Fully Random

We assume that the *host* interconnection network is fully random, i.e., the edges between any compute nodes are established arbitrarily. The resulting graph has no self-loops or parallel edges.

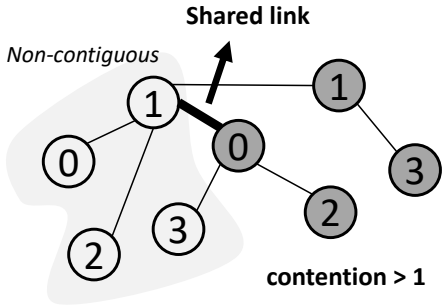
We still investigate two patterns of the topology embedding: regular and random. Since both the job mapping algorithms are similar to the case when the *host* topology is a WS small-world network (Section III-B), the corresponding description is omitted in this section.

IV. Evaluation

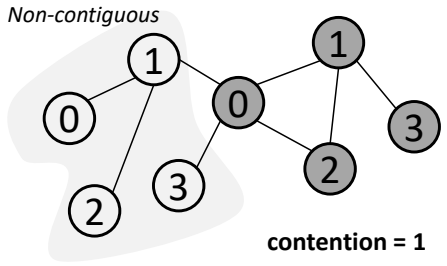
In this section, we first evaluate the execution time of single and multiple NPB applications using regular and random topology embeddings, and the impact of network *contention* between multiple communicating tasks on their execution times. Then, we evaluate the performance of regular and random graph embedding. Finally, we evaluate the job scheduling performance of regular and random topology embeddings using a compound workload of NPB applications.

A. Single Application

We use an event driven simulator SimGrid (v3.12) [25] to simulate the executions of different NPB applications [26], including FT (Fast Fourier), IS (Integer Sort), CG (Conjugate Gradient), BT (Block Tri), SP (Scalar Penta) and MG (Multi-Grid) taken from NAS parallel benchmarks [27], and MM



(a) One link is shared by two parallel jobs.



(b) No link is shared by two parallel jobs.

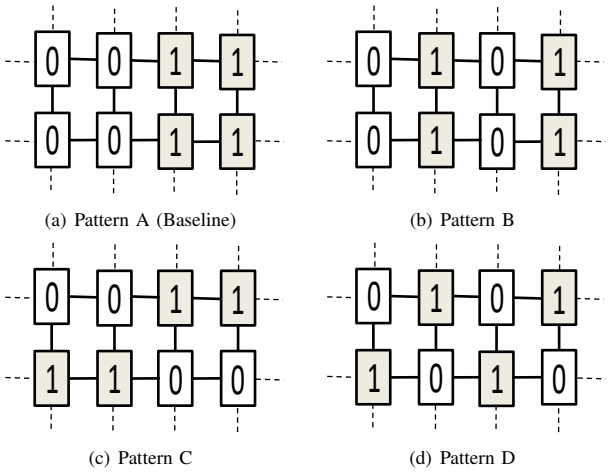
Figure 4. The cases of job mapping on a non-contiguous topology.

(Matrix Multiplication). We get a series of simulation results of execution times (cycles) for these NPB applications running on different *guest* topologies with different job mapping dilations. The execution time of each NPB application is determined by its embedded topology and mapping dilation.

The computation power of each compute node is set to 100GFlops. The cable bandwidth is set to 40Gbps. The switch latency is set to 60ns and the switch bandwidth is set to 10Pbps. Notice that the execution time order of the benchmark applications in SimGrid is five milliseconds to five seconds.

Table 1 shows the execution times of NPB applications on regular/random 64-node topologies. The regular topologies include 2-D mesh (M(2, 4)), 3-D mesh (M(3, 6)) and 3-D torus (T(3, 6)), and the random topologies include degree-4 (R(4, d)) and degree-6 (R(6, d)) topologies with different mapping dilations ($d \in [1, 4]$) each. For most of the NPB applications, such as FT, IS, CG and MM, the topology mapping on a degree-6 dilation-1 random topology (R(6, 1)) brings the shortest execution time among all the cases. For instance, the application of MM benefits from the job mapping on random topologies (with any degree and dilation) rather than regular topologies such as 2-D mesh, 3-D mesh and 3-D torus in terms of execution time. This is because a regular job mapping may hurt the communication of node pairs due to large path hops, whereas they can be reduced by a random job mapping for a specific communication pattern. For all the cases of job mapping on random topologies, a larger degree and a smaller dilation can help to decrease the execution time. For BT and SP, the topology mapping on 3-D torus (T(3, 6)) gets the shortest execution time among all the cases; for MG, the topology mapping on 2-D mesh (M(2, 4)) is the winner.

Overall, the job mapping on a regular topology does not suit each application's communication pattern between communicating tasks. The job mapping on a random topology with a larger degree and a smaller dilation seems to be the best choice for most of the applications.



(a) Pattern A (Baseline) (b) Pattern B

(c) Pattern C (d) Pattern D

Figure 5. Simultaneous executions of two parallel jobs on the system.

B. Multiple Applications

Remember that in a *non-contiguous* random topology embedding, one or more links can be shared by multiple applications. In this case, the bandwidth of the shared link(s) is simultaneously occupied by these applications, and this imposes an impact on the application execution times. We evaluate the average execution time of the conflicting applications if link sharing occurs between them on the system.

We assume that the *host* topology is 2-D (4×2) torus and two parallel applications simultaneously run with several different patterns as shown in Fig. 5. In pattern A (Fig. 5(a)), two parallel jobs can run without link sharing over the same topology. We use pattern A as baseline since two jobs are mapped on a regular 2-D (2×2) mesh topology respectively. Comparatively, in patterns B (Fig. 5(b)), C (Fig. 5(c)) and D (Fig. 5(d)), one or multiple links are shared by the two jobs, thus their execution times may be influenced by the link sharing.

The computation power of each compute node is set to 1GFlops. The cable bandwidth is set to 1Gbps and the cable latency is set to 50us. The switch bandwidth is set to 800Mbps.

We first get the time-independent traces of the NPB applications by SimGrid. We use Batsim [28] to run multiple NPB applications in parallel in replay mode and evaluate the average execution time of two NPB applications over a 2-D (4×2) torus topology. Overall, as shown in Fig. 6, the topology embedding with pattern A behaves the best among all the cases in terms of execution time. Because two jobs are both mapped on regular 2-D (2×2) mesh topologies, there is no link shared by these two jobs. In this case, each job can separately make the best use of bandwidth capability of any link over the *host* topology. The average execution time is increased when using pattern B, C or D. Among all the NPB applications, LU is the most vulnerable one to the network *contention* against another simultaneously running application. It has a 104.1% rise in execution time when using pattern C compared to pattern A. The application of FT is relatively immune to the network *contention*, because it leads to only

Table 1. Execution time (s) of NPB applications on regular/random 64-node topologies (the best/worst result is shown in green/yellow).

NPB	Mesh/Torus(dimension, degree)			Random(degree, dilation)							
App.	M(2, 4)	M(3, 6)	T(3, 6)	R(4, 1)	R(4, 2)	R(4, 3)	R(4, 4)	R(6, 1)	R(6, 2)	R(6, 3)	R(6, 4)
FT	0.0298	0.0200	0.0296	0.0149	0.0151	0.0154	0.0158	0.0125	0.0128	0.0131	0.0135
IS	0.0123	0.0105	0.0120	0.0063	0.0063	0.0064	0.0064	0.0055	0.0055	0.0056	0.0056
CG	1.1700	1.0875	1.2793	1.0155	1.0234	1.0299	1.0361	0.9446	0.9510	0.9575	0.9639
BT	1.2038	0.3817	0.2741	0.5326	0.5344	0.5367	0.5405	0.4167	0.4201	0.4242	0.4299
SP	1.9574	0.6189	0.4629	0.8597	0.8651	0.8708	0.8769	0.6779	0.6832	0.6889	0.6958
MG	0.0179	0.0285	0.0275	0.0266	0.0270	0.0278	0.0287	0.0234	0.0241	0.0247	0.0254
MM	0.4266	0.2932	0.2713	0.2063	0.2051	0.2059	0.2059	0.1855	0.1881	0.1865	0.1872

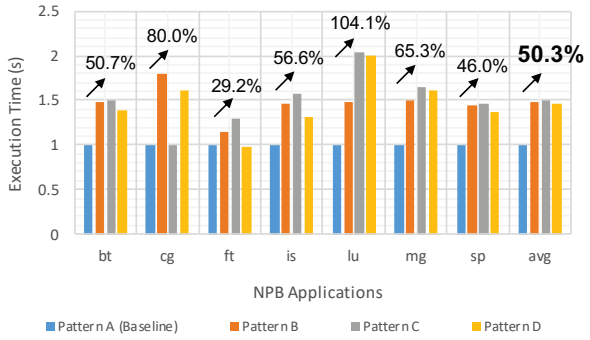


Figure 6. Average execution time of two simultaneously running NPB applications.

a 29.2% increase in execution time when using pattern C compared to pattern A.

C. Topology Embedding

We investigate the performance of graph embedding when mapping jobs on regular and random topologies. We take two metrics as our evaluation objectives: average shortest path length (ASPL) and average diameter. The former refers to the average shortest communication path hops between the compute nodes in a graph embedding, and the latter refers to the average largest communication path hops among all the compute node pairs in a graph embedding.

We develop an event-driven simulator in Python 2.7 to model job mapping and run the simulation in a machine with Intel i7 CPU and 4GiB Memory. We use a common approach to model parallel "rigid" jobs [29], which refer to the jobs that use a fixed number of resources during runtime. We generate $n = 2000$ jobs as a workload with random arrival timings for a *Poisson* process with $\lambda = \frac{n}{1000}$. We use a workload composed of different NPB applications, and each job specifies the required number (4, 16, 64 or 64) of compute nodes.

The *host* topology is assumed to be a 3-D ($16 \times 16 \times 12$) torus, a WS small-world network (partially random topology) and a fully random topology, respectively. For fair comparison, all the *host* topologies have 3072 compute nodes with the same degree of 6. The probability of rewiring each edge in the WS small-world network is $p = 10\%$.

When we consider job mapping on random topologies, each job is still prioritized to be mapped on a regular topology. Only when current regular topology mapping cannot be found, a random topology embedding is considered.

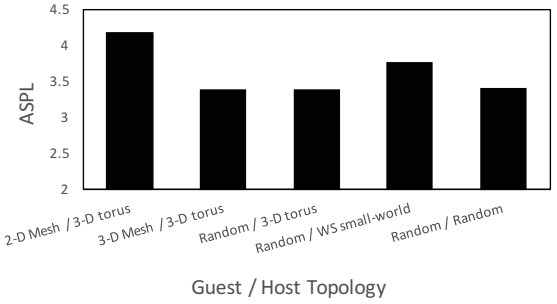


Figure 7. ASPL of embedded topologies.

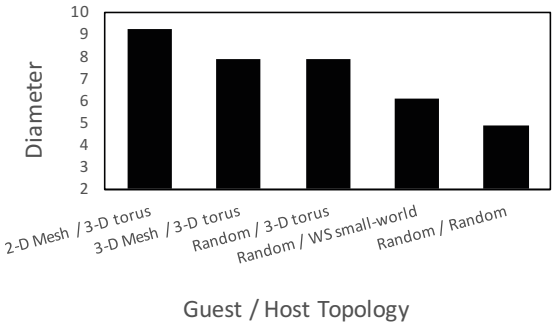
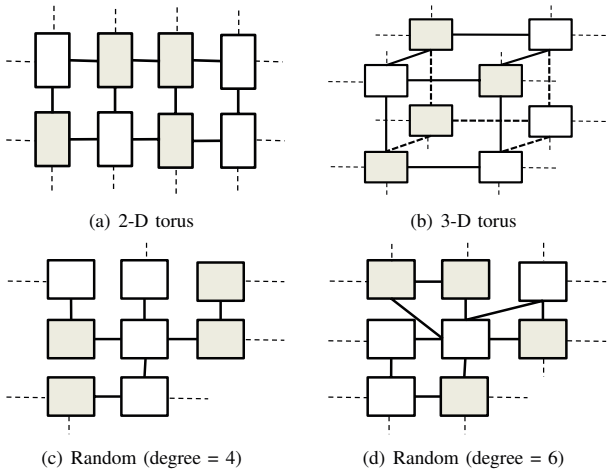


Figure 8. Average diameter of embedded topologies.

As shown in Fig. 7, when the *host* topology is 3-D torus, the random graph embedding obtains a smaller ASPL than that on 2-D mesh and a close ASPL to that on 3-D mesh. When the *guest* topology is random, a fully random *host* topology presents a certain advantage over a WS small-world *host* network.

Similarly, as shown in Fig. 8, when the *host* topology is 3-D torus, the random graph embedding obtains a smaller diameter than that on 2-D mesh and a close diameter to that on 3-D mesh. Interestingly, the diameter obtained by a random graph embedding varies from different *host* topologies. Compared to 3-D torus, when the *host* topology is a WS small-world (partially random) network, the diameter of the random graph embedding becomes smaller, and when the *host* topology is fully random, it is even much smaller.

Notice that, the job mapping on a *non-contiguous* random topology leads to an unconnected graph embedding, thus



(a) 2-D torus (b) 3-D torus

(c) Random (degree = 4) (d) Random (degree = 6)

Figure 9. The example of random job mapping over different host topologies in the evaluation of job scheduling.

we do not show its evaluation results regarding ASPL and diameter.

D. Job Scheduling

To evaluate the job scheduling performance, we take into consideration the bandwidth limitation of the links simultaneously occupied by multiple jobs. That is, when the link sharing occurs on the system, the job execution time may be influenced by another simultaneously running application sharing the same link with limited bandwidth. If each job is mapped on randomly selected compute nodes, there is a high possibility that one or more links are shared by multiple jobs. In this section, we investigate the impact of random job mapping on job scheduling performance.

We first introduce some definitions and notations. Assume that job j out of all submitted jobs J is scheduled on n_j nodes out of a system of size N. Its execution time is t_j during arrival timing a_j and completion timing c_j. The turnaround time of job j is $c_j - a_j$, and can be divided into wait time and execution time. The former refers to the time spent waiting in job queue, and the latter refers to the time spent executing that job.

We use Batsim as the job scheduler to evaluate the job scheduling performance of random job mapping over several *host* topologies, including 2-D torus, 3-D torus and random topologies (with degree-4 and degree-6), as shown in Fig. 9. For fair comparison and realistic simulation time, all the *host* topologies have 64 compute nodes. The computation power of each compute node is set to 1GFlops. The cable bandwidth is set to 1Gbps and the cable latency is set to 50us. The switch bandwidth is set to 800Mbps.

Figure 10 shows the model of job scheduling in our evaluation. We generate $n = [20, 200]$ jobs as a workload with random arrival timings for a *Poisson* process with $\lambda = \frac{n}{20}$. We use a workload composed of different NPB applications, and each job specifies the required number (4 or 16) of compute nodes. We use first-come-first-serve (FCFS) as the job queuing policy and EASY backfilling [30] as the job scheduling strategy.

Figure 11 presents the makespan of the whole job scheduling process over different *host* topologies. Among all the

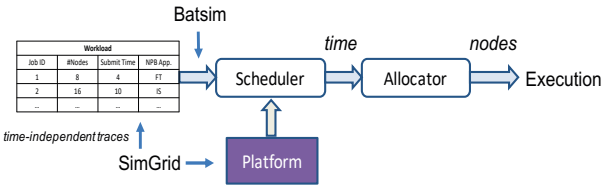


Figure 10. The model of job scheduling in our evaluation.

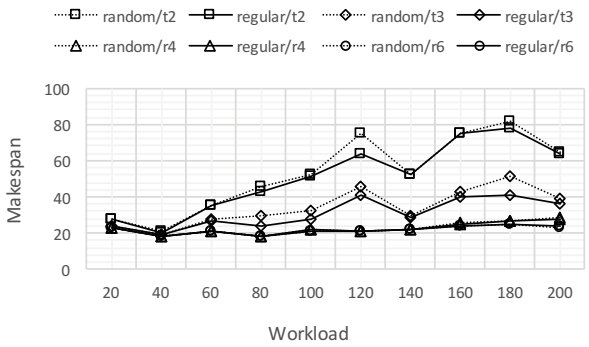


Figure 11. Makespan (guest/host topology, t2: 2-D torus, t3: 3-D torus, r4: random with degree-4, r6: random with degree-6).

host topologies, 2-D torus (t2) causes the worst performance because it leads to the longest makespan. The *host* topology of 3-D torus (t3) brings shorter makespan than 2-D torus (t2). The job mapping on a regular topology marginally outperforms that on a random topology over 2-D torus (t2) and 3-D torus (t3). This is because a regular topology mapping results in a *contiguous* graph embedding, which brings shorter execution time of the mapped job. In this case, although a random topology mapping can have shorter wait time, it still can not compensate for the performance loss due to larger execution time. Fortunately, over 2-/3-D torus the difference between a regular topology mapping and a random topology mapping is really minor. Considering the simplicity and system utilization, the little punishment is acceptable for the random topology mapping over 2-/3-D torus.

A similar tendency can be seen from Fig. 12, which shows the average turnaround time of all dispatched jobs over different *host* topologies. The *host* topology of 3-D torus (t3) brings shorter average turnaround time than 2-D torus (t2). Still, the job mapping on a regular topology marginally outperforms that on a random topology over 2-/3-D torus.

Overall, the random *host* topologies show the best performance in terms of makespan and turnaround time. The advantages become obvious when the workload is larger than 40. For example, when the workload is 200, the random job mapping over a random topology with degree-6 (r6) reduces 64% of makespan and 80% of turnaround time when compared with the regular job mapping over 2-D torus, and reduces 36% of makespan and 54% of turnaround time when compared with the regular job mapping over 3-D torus. When the *host* topology is random with degree-4 (r4) or degree-6 (r6), there is little difference between a regular topology embedding and a random topology embedding, and the latter even shows tiny advantage over the former. Moreover, a random *host* topology with degree-4 (r4) seems no much difference with degree-6

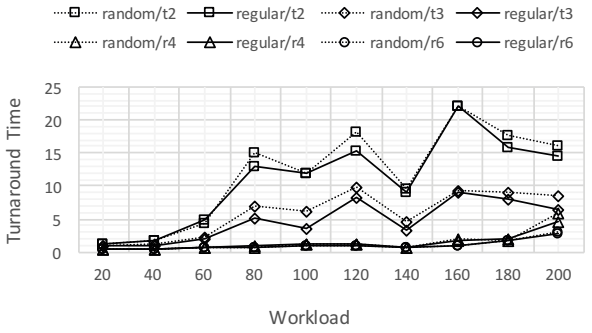


Figure 12. Average turnaround time (guest/host topology, t2: 2-D torus, t3: 3-D torus, r4: random with degree-4, r6: random with degree-6).

(r6). In other words, the degree of a random *host* topology has a very limited impact on the job scheduling performance.

V. Conclusion

Efficient mapping of application to network topology gains importance as HPC systems grow to petascale and beyond. Current supercomputers and datacenters assign compute nodes connected by a regular *contiguous* topology, while a random topology can accept any type of *guest* topologies for user jobs by connecting even *non-contiguous* compute nodes. In this context, instead of conventional job mapping on regular topologies, we analyzed the impact of network *contention* on simultaneously running applications, and explored the job mapping on random topologies for the purpose of improving job scheduling performance.

By event-driven simulations we showed that, compared to the *guest* topology, the *host* topology has a more significant impact on job scheduling performance. Interestingly, the job scheduling performance benefits more from a random *host* topology than from a regular *host* topology such as torus. In this case, the job mapping on random *guest* topologies performs comparably to or even better than that on regular *guest* topologies. Therefore, the random job mapping can present the best job scheduling performance among all the cases. This avoids optimization of a given user job to a specified network topology, and shows a great long-term potential for supporting parallel user applications in job scheduling abilities.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number 19K20263.

References

[1] D. K. Dror G. Feitelsona, Dan Tsafrirb, "Experience with using the Parallel Workloads Archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, October 2014.

[2] O. Tuncer, V. J. Leung, and A. K. Coskun, "Pacmap: Topology mapping of unstructured communication patterns onto non-contiguous allocations," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, 2015, pp. 37–46.

[3] [Online]. Available: https://en.wikipedia.org/wiki/Cray_XT6

[4] [Online]. Available: https://ja.wikipedia.org/wiki/Blue_Gene

[5] A. Bhatele and L. V. Kalé, "Application-specific topology-aware mapping for three dimensional topologies," *2008 IEEE International Symposium on Parallel and Distributed Processing*, pp. 1–8, 2008.

[6] Y. Hu, I. Fujiwara, and M. Koibuchi, "HPC Job Mapping over Reconfigurable Wireless Links," in *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2016)*, May 2016, pp. 570–575.

[7] Y. Georgiou, E. Jeannot, G. Mercier, and A. Villiermet, "Topology-aware job mapping," *The International Journal of High Performance Computing Applications*, vol. 32, no. 1, pp. 14–27, 2018. [Online]. Available: https://doi.org/10.1177/1094342017727061

[8] J. Navaridas, J. Miguel-Alonso, F. J. Ridruejo, and W. Denzel, "Reducing complexity in tree-like computer interconnection networks," *Parallel Comput.*, vol. 36, no. 2-3, pp. 71–85, Feb. 2010. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2009.12.004

[9] Y. Hu and M. Koibuchi, "The impact of job mapping on random network topology," 11 2018, pp. 79–85.

[10] M. Koibuchi, H. Matsutani, H. Amano, D. F. Hsu, and H. Casanova, "A Case for Random Shortcut Topologies for HPC Interconnects," in *Proc. of the International Symposium on Computer Architecture (ISCA)*, 2012, pp. 177–188.

[11] S. Moh, C. Yu, H. Youn, K. Lee, and D. Han, "Mapping strategies for switch-based cluster systems of irregular topology," in *Proceedings of the Internatoinal Conference on Parallel and Distributed Systems - ICPADS*, 02 2001, pp. 733 – 740.

[12] A. Rosenberg, *Issues in the study of graph embeddings*, wg 1980 ed., ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 1981, vol. 100.

[13] G. Fox, V. Getov, L. Grandinetti, G. Joubert, and T. Sterling, *New Frontiers in High Performance Computing and Big Data*. IOS Press, 2017, vol. 30 of Advances in Parallel Computing.

[14] J. T. Vogelstein, J. M. Conroy, V. Lyzinski, L. J. Podrazik, S. G. Kratzer, E. T. Harley, D. E. Fishkind, R. J. Vogelstein, and C. E. Priebe, "Fast approximate quadratic programming for graph matching," *PLOS ONE*, vol. 10, no. 4, pp. 1–17, 04 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0121002

[15] T. Agarwal, A. Sharma, A. Laxmikant, and L. V. Kale, "Topology-aware task mapping for reducing communication contention on large parallel machines," in *Proceedings 20th IEEE International Parallel Distributed Processing Symposium*, April 2006, pp. 10 pp.–.

[16] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proceedings of the International Conference on Supercomputing*, ser. ICS '11. New York, NY, USA: ACM, 2011, pp. 75–84. [Online]. Available: http://doi.acm.org/10.1145/1995896.1995909

[17] G. Lee, N. Tolia, P. Ranganathan, and R. H. Katz, "Topology-aware resource allocation for data-intensive workloads," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 1, pp. 120–124, Jan. 2011. [Online]. Available: http://doi.acm.org/10.1145/1925861.1925881

[18] J. Wu, X. Xiong, and Z. Lan, "Hierarchical task mapping for parallel applications on supercomputers," *J. Supercomput.*, vol. 71, no. 5, pp. 1776–1802, May 2015. [Online]. Available: http://dx.doi.org/10.1007/s11227-014-1324-5

[19] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware mpi topology functions," in *Proceedings of the 18th European MPI Users' Group Conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 50–60. [Online]. Available: http://dl.acm.org/citation.cfm?id=2042476.2042484

[20] C. Albing, N. Troullier, S. Whalen, R. Olson, J. Glenski, H. Pritchard, and H. Mills, "Scalable node allocation for improved performance in regular and anisotropic 3d torus supercomputers," in *Recent Advances in the Message Passing Interface*, Y. Cotronis, A. Danalis, D. S. Nikolopoulos, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 61–70.

[21] X. Yang, Z. Zhou, W. Tang, X. Zheng, J. Wang, and Z. Lan, "Balancing job performance with system performance via locality-aware scheduling on torus-connected systems," in *2014 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2014, pp. 140–148.

[22] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, pp. 440–442, 1998.

[23] B. M., *Small Worlds and the Groundbreaking Science of Networks*. W. W. Norton, 2003.

[24] R. P. Ishii, R. F. de Mello, and L. T. Yang, "A complex network-based approach for job scheduling in grid environments," in *HPCC*, 2007.

[25] "Simgrid: Versatile simulation of distributed systems," http://simgrid.gforge.inria.fr/.

[26] The NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/NPB/.

[27] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga, "The nas parallel benchmarks," https://www.nas.nasa.gov/assets/pdf/techreports/1994/rnr-94-007.pdf, March 1994.

[28] [Online]. Available: https://batsim.readthedocs.io/en/latest/index.html#

[29] D. G. Feitelson, *Workload Modeling for Computer Systems Performance Evaluation*. Cambridge University Press, March 2015.

[30] Ward, W. A. Jr., C. L. Mahood, and J. E. West, "Scheduling jobs on parallel systems using a relaxed backfill strategy," in *Proceedings of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, July 2002, pp. 88–102.