Application Mapping and Scheduling of Uncertain Communication Patterns onto Non-Random and Random Network Topologies

Yao HU†a, Nonmember and Michihiro KOIBUCHI†, Member

SUMMARY Due to recent technology progress based on big-data processing, many applications present irregular or unpredictable communication patterns among compute nodes in high-performance computing (HPC) systems. Traditional communication infrastructures, e.g., torus or fat-tree interconnection networks, may not handle well their matchmaking problems with these newly emerging applications. There are already many communication-efficient application mapping algorithms for these typical non-random network topologies, which use nearby compute nodes to reduce the network distances. However, for the above unpredictable communication patterns, it is difficult to efficiently map their applications onto the non-random network topologies. In this context, we recommend using random network topologies as the communication infrastructures, which have drawn increasing attention for the use of HPC interconnects due to their small diameter and average shortest path length (ASPL). We make a comparative study to analyze the impact of application mapping performance on non-random and random network topologies. We propose using topology embedding metrics, i.e., diameter and ASPL, and list several diameter/ASPL-based application mapping algorithms to compare their job scheduling performances, assuming that the communication pattern of each application is unpredictable to the computing system. Evaluation with a large compound application workload shows that, when compared to non-random topologies, random topologies can reduce the average turnaround time up to 39.3% by a random connected mapping method and up to 72.1% by a diameter/ASPL-based mapping algorithm. Moreover, when compared to the baseline topology mapping method, the proposed diameter/ASPL-based topology mapping strategy can reduce up to 48.0% makespan and up to 78.1% average turnaround time, and improve up to 1.9x system utilization over random topologies.

key words: high-performance computing (HPC), interconnection network, topology embedding, job scheduling

1. Introduction

In high-performance computing (HPC) systems, multiple compute nodes that communicate with each other cooperate to simultaneously accommodate a large number of highly parallel running applications [1]. Both scales of compute nodes and applications have been growing exponentially in the last decade [2]. The steadily increasing number of compute nodes calls for more efficient interconnection networks and application mapping strategies. Traditionally, these compute nodes are usually connected by a non-random interconnection network, e.g., a torus [3] or fat-tree [4] topology. In this study, for ease of understanding, we use non-random network topologies to represent the network topologies with one or more regular or symmetric characteristics, which are usually used in commercial supercomputers, such as k-ary n-cubes and hypercubes. For example, the custom supercomputers Cray XT6 [5] and Blue Gene/Q [6] are connected by a 3-D torus network topology and a 5-D torus network topology respectively. Each application is executed on a (small) embedded k-ary n-cube topology on the both supercomputer systems. By dispatching communicating tasks to nearby compute nodes, the network contention and communication latency can be minimized. That is, such topology mapping can benefit from low-latency high-bandwidth communication between nearby compute nodes for traditional applications with regular communication patterns [3].

Due to recent technology progress based on big-data processing, its applications present irregular or unpredictable communication patterns among compute nodes. As a result, existing application mapping algorithms for non-random network topologies do not fit with these diverse applications on the computing system, which can have various communication patterns, from adjacent point-to-point communication within a virtual process grid, to irregular point-to-point communication, to diverse collectives with different group sizes, to any combination of the above [7]. In this case, a traditional non-random network topology itself may have a significant limitation on the performance of the embedded topologies which simultaneously accommodate multiple running applications.

On the other hand, random network topologies have drawn increasing attention for the use of HPC interconnects. A random network topology is generated either by a fully random graph [8] or by adding random links to a baseline network topology so that each node has the same degree [9]. Random network topologies can be built for any pair of the number of nodes and degree. In this study, we consider generating a random network topology by randomly connecting a pair of nodes that have less links than the network degree. We enforce that all the nodes have the same degree if possible. Recent works [9]–[11] have shown that, compared to classical non-random topologies of identical degree, random topologies interestingly improve the topology embedding diameter by up to one order of magnitude. A similar improvement is achieved for the average shortest path length (ASPL). Therefore, random topologies would be recommended in current and future supercomputers for emerging scientific and big-data analysis parallel applications whose communication patterns are unpredictable. This
is because the low-diameter/ASPL topologies, i.e., random topologies, are beneficial for such communication patterns. However, the problem of application mapping over random topologies is intricate owing to the irregular connection and complicated routing [9]. In this study, we investigate the performance of application mapping over random topologies, and compare it with that over non-random topologies.

Figure 1 depicts an example of application mapping over a non-random (2-D mesh) network topology and a random network topology respectively, which have the same degree of four. When the virtual process topology of an application is a regular $2 \times 2$ mesh (blue), both the non-random and random topologies can accommodate the application with an optimum physical processor topology. However, when the virtual process topology of an application is irregular (red), the non-random network topology cannot provide an optimum physical processor topology due to a regular interconnect. Comparatively, the random network topology can offer the application an optimum physical processor topology for its efficient execution. We define an efficient application mapping as a dilation-1 topology embedding, and an inefficient application mapping as a dilation-$x$ topology embedding. Dilation in the topology embedding refers to the length (in number of hops) of the shortest path between two embedded vertices of an edge. Assume a mapping $F$ from the vertices of a graph $G$ to the vertices of a (larger) graph $H$. Given an edge in $G$ between two vertices $v$ and $w$, its dilation is the length of the shortest path between $F(v)$ and $F(w)$ in number of hops in $H$. The dilation for a mapping is the maximum value of all the edge dilations. For example, a mapping with dilation = 1 does not expand any edge of $G$ in $H$, and a mapping with dilation = 2 leads to the maximum 2 hops for any edge of $G$ in $H$.

As the scale of HPC systems grows to petascale and beyond, it gains unprecedented importance for developers to design efficient algorithms for mapping of application to network topology. Many application mapping schemes have been proposed based on the existing systems of non-random network topologies [3], [12]–[15]. Some of them are designed for specific applications or networks, and the others require significant effort by developers or users to successfully apply them. In this study, we propose a new application mapping strategy which turns attention to limitation on the diameter and ASPL of topology embedding. We list several diameter/ASPL-based application mapping algorithms and evaluate their job scheduling performances over non-random and random network topologies. One main objective of this work is to show that a large workload of diverse applications can be better supported by using the proposed diameter/ASPL-based mapping strategy with a certain time-space tradeoff, i.e., we expect to improve the system utilization and reduce the application response time.

We observe from our previous work [16] that, 1) given the same network degree, different random topologies can obtain very close performance in terms of execution time of typical parallel applications, and 2) a smaller dilation (i.e., diameter or ASPL) of the embedded topology can bring a slightly shorter execution time of typical parallel applications. For example, for the application of Matrix Multiplication (MM), a dilation-4 random topology takes $1.014x$ execution time compared to a dilation-1 random topology when their network degrees are both 6. We thus consider that the performance fluctuation can be totally acceptable if the same network degree and close diameters or ASPLs are specified for different random topologies. Random topologies may be not suitable for some traditional explicit workloads. However, as reported in the work [11], random topologies outperform torus topologies of identical degree by 18% on average in terms of application execution, although the behavior performance strongly depends on the benchmark. For instance, random topologies outperform torus topologies by 128% for the execution of Graph500, while having 15% degradation for the execution of LU taken from NAS Parallel Benchmarks [17]. Thus, it would be very expensive to design and adopt a perfectly suitable topology for running each specific application when its communication pattern is unpredictable beforehand. Under this premise, we claim that the proposed mapping strategy by restricting embedding diameter or ASPL can be beneficial for such applications. Furthermore, the performance of diameter or ASPL can be improved using random topologies by up to one order of magnitude when compared to non-random topologies of identical degree [9]. We thus consider that, low-diameter or low-ASPL random topologies would be more beneficial for the applications of unknown communication patterns.

A preliminary version of this paper appeared in Ref. [18]. Compared to that work, this version includes a mechanism for mapping applications over non-random network topologies with restriction on the maximum diameter or ASPL. It also provides a comparative evaluation of non-random and random topologies by using the same diameter/ASPL-based application mapping strategy. Finally, the graph embedding metrics are evaluated assuming the same degree of non-random and random topologies.

Our main contributions in this work are as follows.

- We proposed using random network topologies instead of existing non-random network topologies for current emerging applications with irregular or unpredictable
communication patterns.

- We proposed an application mapping strategy which places restrictions on the maximum diameter and ASPL of topology embedding. We made a comparative study to evaluate several diameter/ASPL-based topology mapping algorithms over non-random and random network topologies.
- By a series of event-driven simulations with a compound application workload, we showed that, compared to non-random topologies, random topologies can reduce the average turnaround time up to 39.3% by a random connected mapping method and up to 72.1% by a diameter/ASPL-based mapping algorithm.
- Through evaluation we showed that, compared to the baseline topology mapping method, the proposed application mapping strategy improves the job scheduling performance by reducing the makespan up to 48.0% and the average turnaround time up to 78.1% and improving up to 1.9x system utilization over random network topologies.

The remainder of this paper is organized as follows. Section 2 discusses background information and related works of this paper. In Sect. 3, we analyze the proposed application mapping strategy. In Sect. 4, we present our experimental setup and evaluation on performance of application mapping over non-random and random network topologies. We conclude this work in Sect. 5.

2. Background and Related Work

2.1 Application Mapping Problem

Application mapping can be regarded as an equivalent to topology embedding in the field of graph theory [19], we thus employ a notation that extends that used for topology mapping in this study.

We use a graph $C = (V_c, E_c)$ to represent the logical inter-process communication pattern, 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 $v \in V_c$. If isolated vertices exist, the graph $C$ is disconnected. In this case, it represents the concurrent execution of multiple unrelated communicating entities. If process $u$ does not communicate directly with process $v$, the value of $E_c(u, v)$ becomes zero. In this study, we take into consideration only the connected graph $C$ for a single application and regard the disconnected graph $C$ as multiple unrelated parallel applications.

Similarly, we use a graph $G = (V_g, E_g)$ to represent the physical inter-processor interconnection network, where $V_g$ is the set of physical nodes, and $E_g(m, n)$ is the bandwidth capacity of the link connecting node $m \in V_g$ to node $n \in V_g$. If node $m$ and node $n$ are not directly connected by any link, the value of $E_g(m, n)$ is zero. The communication from node $m$ to node $n$ usually travels via the shortest path between the node pair. If each link of one path occurs at most once, the path is regarded as a simple path. The set of simple paths that connect node $m$ to node $n$ is represented by $P(m, n)$.

Thus, a topology mapping can be identified by mapping the vertices of $C$ (i.e., processes) to the vertices of $G$ (i.e., nodes or processors), which is represented by a function $F$: $V_c \rightarrow V_g$. In this work, we assume that the communication patterns are unknown beforehand, however we want to mitigate the impact of link contention [20] between different applications on the bandwidth occupied by any specific application, i.e., the traffic via path $p \in P(m, n)$ should be exclusively occupied by a single application and not shared by multiple simultaneously running applications. Therefore, any embedded topology $G$ is assumed to be a connected graph.

2.2 Topology-Specific Application Mapping

The application mapping problem in most of recent supercomputers is constrained to that of mapping smaller mesh or torus network topologies over larger mesh or torus network topologies [22]. It has been shown that due to node fragmentation by the traditional mapping on regular topologies their system utilizations are usually low. For instance, the Blue Waters system [21] employs the Cray Gemini interconnect, which implements a 3-D $(24 \times 24 \times 24)$ torus topology. Figure 2 shows the utilization of compute nodes on the Blue Waters system at 19:29, October 10th, 2018. It is obvious that there are many cases where a large number of unused compute nodes exist.

There have been also dozens of application mapping strategies which are designed for specific network topologies. For instance, Rashidi et al. [23] introduced an application mapping strategy for both the intra and inter node physical topologies to the whole parallel computing system by using a weighted graph model. They used some application sets to produce some interesting evaluation results, which however are not tested with real workload traces or integrated with real resource and job management systems (RJMSs). Jingjin Wu et al. [24] used generic recursive algorithms and proposed a hierarchical application mapping algorithm for both the torus and fat-tree network topologies of modern supercomputers. In the evaluation, they showed good performance of the proposal with low

![Fig. 2](image-url) The utilization of compute nodes on the Blue Waters system [21] at 19:29, October 10th, 2018.
overhead. Yang et al. [25] showed a window-based locality-aware job scheduling method only for the torus topology. They made use of the extension of the algorithms solving the 0-1 Multiple Knapsack problem for resource allocation in order to preserve node contiguity and scheduled multiple applications at the same time. Their objective is to optimize the application execution as well as the whole system performance. Albing. Carl et al.[26] also targeted the torus network topology and presented a locality-aware application mapping study by using space filling curve to order processors and map the compute nodes onto a one-dimensional list.

In this study, besides the above traditional non-random topologies, random topologies are also assumed to be our interconnection networks for application allocation. For fair comparison, we use the same diameter/ASPL-based application mapping strategy over both non-random and random topologies.

2.3 Application-Specific Application Mapping

Several application mapping schemes for specific HPC applications have been proposed over the years. For example, Vogelstein et al. proposed the fast approximate quadratic (FAQ) assignment algorithm[27], which however is too expensive for many large-scale HPC problems in terms of computation cost. Agarwal et al. proposed an algorithm [28] designed for mapping and load balancing of parallel applications, which has a similar complexity compared to FAQ. Hoefler et al. proposed three general algorithms[4] 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. Besides, the work of TARA[29] used a description of application to allocate compute resources, which is tailored for a very specific class of applications.

In our case, we do not use any calculation of bandwidth or latency for resource allocation on the system. Instead, we design general-purpose diameter/ASPL-based topology mapping algorithms over both non-random and random topologies. Our topology mapping strategy simply takes into account the availability of local compute nodes and thus is compatible with diverse application behaviors.

3. Diameter/ASPL-Based Application Mapping

3.1 Concept

In this work, we assume that the communication pattern of the application is uncertain or unknown to a job mapper, so that the proposed application mapping algorithms are compatible with any type of applications. The host network topology is defined as the entire interconnection network of the target system, and a guest network topology is defined as an embedded topology assigned to an incoming application.
accommodate a larger number of applications over the same random topology. Table 1 shows the notations used in the following description of our diameter/ASPL-based application mapping strategy and its job scheduling performance evaluation.

3.2 Lower Bound of Diameter/ASPL

We first calculate the lower bound of the diameter and ASPL given the number of nodes \( n \) and the network degree \( d \). According to the Moore Bound [30] in the graph study, the lower bound of the diameter of the constructed graphs is denoted by \( K_{n,d} \), which is calculated as follows.

\[
K_{n,d} = \begin{cases} \left\lfloor \frac{(n-1)}{2} \right\rfloor & \text{if } d = 2 \\ \left\lfloor \log_{d-1} \left( \frac{(n-1)d-2}{d} + 1 \right) \right\rfloor & \text{if } d > 2 \end{cases}
\]

Likewise, the lower bound of the ASPL of the constructed graphs is denoted by \( L_{n,d} \), which is calculated as follows [31].

\[
L_{n,d} = \begin{cases} 1 & \text{if } K_{n,d} = 1 \\ S_n^d + R_{n,d} & \text{if } K_{n,d} \geq 2 \\ \frac{K_{n,d} - 1}{n-1} & \text{if } K_{n,d} = 1 \end{cases}
\]

where \( S_n^d = \sum_{i=1}^{d(n-1)} i(d-1)^{i-1} \) and \( R_{n,d} = n - 1 - \sum_{i=1}^{K_{n,d} - 1} d(d-1)^{i-1} \).

Based on the above two equations, given the number of nodes \( n \) and the network degree \( d \), we can figure out the lower bound of the diameter and ASPL of any possible embedded guest topology.

3.3 Topology Mapping Strategy

Assume that an incoming application requires \( n \) (\( n \leq N \)) compute nodes for its execution. In our approach, a guest topology used for application execution is generated based on the required number of compute nodes \( n \) and the maximum value of diameter \( k \) or ASPL \( l \). Obviously, \( k \geq K_{n,d} \) and \( l \geq L_{n,d} \). To find such an embedded topology, we construct a corresponding hierarchical tree topology (HTT) according to the current node availability over the host topology of a certain degree (\( d = 4 \) in the example as shown in Fig. 5).

According to our previous work [16], we made two observations as follows. First, a smaller diameter or ASPL of the embedded topology can bring a slightly shorter execution time of typical parallel applications. Second, a random (embedded) topology can outperform a non-random (embedded) topology of identical degree, e.g., mesh and torus, in terms of execution time of typical parallel applications.
We thus use HTT as the same mapping algorithm to find a suitable embedded topology of a restrained diameter or ASPL over both non-random and random topologies.

In this section, we explain how to construct such an HTT for topology mapping in our approach. In the following description, if a compute node is not used or occupied by any application, it is tagged as available; otherwise, it is tagged as not available.

### 3.3.1 Root Node

First of all, we determine the root node of an HTT. We categorize all the compute nodes into two groups: $N_n$ and $N_a$. In the beginning, the node set $N_n$ contains all the nodes in the network and the node set $N_a$ is empty. We then perform an ergodic query within the node set $N_n$ to search an available node that has the maximum number of available neighboring nodes, which will be deleted from $N_n$ and then inserted into $N_a$. The first element of $N_a$, which connects the maximum number of available neighboring nodes, is chosen as the root node.

As shown in Fig. 5, the node having 4 available neighboring nodes is chosen as the root node (red) in that for any other node in the network the number of available neighboring nodes is smaller than 4. If there are multiple nodes that have the same maximum number of available neighboring nodes, we randomly choose one of them as the root node temporarily and mark the others as alternative ones for future use. When a required guest topology cannot be found starting from the chosen root node, the alternative ones will be consequently used as the root node to construct a required HTT.

The depth of an HTT is directly related to its diameter or ASPL. The reason why we choose the node having the maximum number of available neighboring nodes as the root node is that we intend to keep the constructed HTT as flat as possible, whose diameter or ASPL can be low. Otherwise, the diameter or ASPL of the generated guest topology may increase as the HTT depth becomes large. Notice that, when the host topology is not random, the choice of the root node becomes easier because each node in the network usually has the same degree. In this case, if all the neighboring nodes are available, the current node can be chosen as an alternative root node.

### 3.3.2 Topology Construction

After the root node (the first element in $N_n$) is chosen, we determine the other available nodes from $N_n$ to construct an HTT for topology embedding as follows. Concretely, we perform an ergodic query within the neighboring nodes of the root node to search available nodes in $N_n$. In the same way, the found available nodes are deleted from $N_n$ and inserted into $N_a$. At this time, however, if the node element newly added to $N_a$ causes an increase of the HTT diameter to exceed $k$, the algorithm goes back to the beginning to choose another alternative root node to perform the above procedure again to construct a required new HTT if it exists.

After performing such a recursive categorization, the whole algorithm stops when the number of elements in $N_a$ is equal to $n$. In this way, as shown in Fig. 5, the resulting contiguous network is a tree-like guest topology. Note that, the HTT construction is performed via breadth-first-search (BFS) to ensure that the resulting HTT keeps flat and low depth so that the generated guest topology has low diameter or ASPL. In the example as shown in Fig. 6, node 6 is connected to both nodes 3 and 4 in a random topology. During the HTT construction, however, due to the BFS procedure of node traversal, node 6 is placed at the second layer to attach node 3, rather than placed at the third layer to attach node 4. In the former case the HTT diameter is 4, whereas in the latter case the HTT diameter becomes 5.

### 3.4 Mapping Criteria

During topology mapping, we decide whether an HTT is accepted as a guest topology for application execution during its construction process. We can apply different mapping criteria based on the limitation on the maximum diameter ($k$) or ASPL ($l$) of topology embedding. In this evaluation, we mainly adopt the following two types of criteria for the decision.

![Fig. 6 The HTT construction is performed via breadth-first-search (BFS) during node traversal. In this example, node 6 is connected to both nodes 3 and 4 in a random topology. During the HTT construction, node 6 is placed at the second layer to attach node 3, rather than placed at the third layer to attach node 4. In the former case the HTT diameter is 4, whereas in the latter case the HTT diameter becomes 5.](image-url)
Algorithm 1 Application mapping with limitation on maximum diameter and ASPL of an embedded topology.

1: procedure map(Topo host, App app, Max_Diameter k, Max_ASPL m)
2:  n ← num_nodes(app) // required number of nodes
3:  Nd ← {} // initiate set of alternative root nodes
4:  Na ← {} // initiate set of available nodes
5:  Nn ← node_set(host) // initiate set of unavailable nodes
6:  index ← 0 // index of current node in Nn
7:  add the nodes that have the maximum number of available neighboring nodes to Nn
8:  for r_node in Nd do // construct HTT from root node
9:      shift r_node from Nn to Na
10:     while 0 < count(Nn) < n do
11:         c_node ← Nn[index] // get current node
12:          for n_node in c_node.neighbors() do
13:              if n_node is available in Nd then
14:                 shift n_node from Nd to Nn
15:                     if calc_diameter(Nn) > k then
16:                         initiate Nn, Nn and break
17:                     end if
18:             end if
19:         end for
20:     end while
21:     index ← index + 1 // next leaf node
22: end procedure

3.4.1 Diameter-Based Solution

The first criterion adopts a diameter-based algorithm, by which we only make a restriction on the maximum diameter \( k \) of a constructed HTT. First, according to the Eq. (1), we get the lower bound of the diameter \( K_{n,d} \) of a potential guest topology given the requested number of compute nodes \( n \) and its network degree \( d \) which is equal to the number of available neighboring nodes of the root node. The calculated \( K_{n,d} \) is used as the limitation of the diameter of a generated HTT, i.e., the diameter of a generated HTT should be not larger than \( K_{n,d} \). Then, in this case, when a traversal node is added to the constructed HTT to make its diameter larger than \( K_{n,d} \), this node will be discarded and the algorithm will go back to choose another alternative root node to construct a new HTT if it exists. This is because adding any other leaf node of the constructed HTT will also increase its diameter to exceed \( K_{n,d} \).

Because an HTT is a tree-like topology, the calculation of its diameter can be done via the node traversal procedure. Obviously, the diameter of an HTT depends on the two leaf nodes with the deepest and second deepest layers stemming from a common parent node. In other words, the sum of the depths of the two leaf nodes stemming from the common parent node determines the HTT diameter. As shown in Fig. 7, the two branches (purple and green) of the root node determines the HTT diameter, which thus is equal to the sum of the depths of the two leaf nodes, i.e., \( L + (L − 1) = 2L − 1 \).

3.4.2 ASPL-Based Solution

The second criterion adopts an ASPL-based algorithm, by which we care more about the ASPL of a constructed HTT rather than its diameter. In other words, a constructed HTT can be accepted only if its ASPL is adequate even though its diameter may be larger than \( K_{n,d} \). First, likewise, according to the Eq. (2), we get the lower bound of the ASPL \( L_{n,d} \) of a potential guest topology given the requested number of compute nodes \( n \) and its network degree \( d \) which is equal to the number of available neighboring nodes of the root node. Then, we set different thresholds for the maximum ASPL of the topology embedding, e.g., 25%, 50%, 75% or 100%, which reflects the maximum possible deviation from the lower bound of ASPL \( L_{n,d} \). That is, if the ASPL of a constructed HTT exceeds \( L_{n,d} \) up to 25%, 50%, 75% or 100%, it can be accepted as a required guest topology. Otherwise, it is discarded and the algorithm returns to choose another alternative root node to construct a new HTT if it exists.

Besides, we investigate the following two different methods of the limitation on the ASPL of the topology mapping in our evaluation.

- **Strict (S-t):** a constructed HTT is rejected once its ASPL exceeds \( t \) of the lower bound \( L_{n,d} \) (0 < \( t \) ≤ 1)
- **Variable (V-t):** the threshold of the maximum ASPL incrementally increases in steps of 25% if a required HTT cannot be found, and eventually its value should not exceed \( t \) of the lower bound \( L_{n,d} \) (0 < \( t \) ≤ 1)
3.5 Algorithm Analysis

Generally, the entire loose application mapping algorithm is divided into three steps. In the first step, all the \( N \) compute nodes in the network are traversed to find the alternative root nodes for HTT construction. The time complexity is simply \( O(N) \). In the next step, starting from the root node, the algorithm checks the availability of the neighboring nodes of current node, whose degree is not greater than the network degree, until the remaining required \( n - 1 \) compute nodes are found for application mapping. The time complexity is \( O(n) \) in this step. Because \( n \) is required to be not greater than \( N \), hence the overall time complexity is \( O(N) \). The third step is to determine whether a constructed HTT is accepted as a required guest topology for application execution. Due to the characteristics of an HTT, the diameter calculation is done according to the depth of the leaf node via the traversal procedure, thus the time complexity of the \( d \)-\( ASPL \)-based application mapping algorithms is just \( O(N) \).

For the \( ASPL \)-based application mapping algorithms, when one node is added to the constructed HTT, its shortest path length towards the other nodes should be calculated, which takes \( O(e + n \log n) \) time when using Fibonacci Heap, e.g., in the Dijkstra’s algorithm, where \( e \) is the number of links in the HTT. Because the number of nodes is one larger than the number of links in the HTT, i.e., \( e = n - 1 \), hence the time complexity of the \( ASPL \)-based application mapping algorithms is \( O(N + n \log n) \).

Obviously, the proposed application mapping algorithms prioritize low-\( d \)-\( ASPL \) topology embedding so as to have low communication latency between compute nodes. If such required topology embedding cannot be found, we can incrementally relax the limitation on the maximum \( d \)-\( ASPL \) for topology embedding.

In the next section, we investigate the impact of different application mapping algorithms on the performance of job scheduling. We interchangeably use the common terms “job” and “application” when referring to their mapping or scheduling in the following description.

4. Evaluation

4.1 Simulation Setup

We developed an event-driven HPC simulator written in Python 2.7 in a machine with Intel i7-6500U (2.50GHz) CPU and 16GiB Memory to model application mapping over non-random and random network topologies. To investigate the effect of the proposed \( d \)-\( ASPL \)-based application mapping strategy, we implement several different algorithms including the following ones, where the communication pattern of each application is assumed to be unknown beforehand.

- **RC**: a naive method to map an incoming application on any random connected guest topology. The application keeps waiting in the job queue if such a random connected topology of required available compute nodes cannot be found. This method is used as the baseline application mapping method in our evaluation.
- **D-MB**: a \( d \)-\( ASPL \)-based algorithm to map an incoming application on a guest topology with the diameter equal to the Moore Bound \( K_{n,d} \). 
- **D-MB-1**: a \( d \)-\( ASPL \)-based algorithm to map an incoming application on a guest topology with the diameter not greater than the Moore Bound \( K_{n,d} \) plus one.
- **D-MB-2**: a \( d \)-\( ASPL \)-based algorithm to map an incoming application on a guest topology with its diameter not greater than the Moore Bound \( K_{n,d} \) plus two.
- **L-S-0.5**: a strict \( ASPL \)-based algorithm to map an incoming application on a guest topology with the \( ASPL \) not greater than the lower bound \( L_{n,d} \) by up to 50%.
- **L-V-1**: a variable \( ASPL \)-based algorithm to map an incoming application on a guest topology with its \( ASPL \) not greater than the lower bound \( L_{n,d} \) by up to an incrementally increased percentage in steps of 25%. The topology mapping with its \( ASPL \) larger than the lower bound \( L_{n,d} \) by 100% and above will be not accepted.

Obviously, RC shows a mapping method when the \( d \)-\( ASPL \) of an embedded topology does not have any restriction as long as it is connected. In this case, RC can minimize the wait time of an incoming application to accelerate its mapping over non-random and random topologies, although its execution time may increase due to a large \( d \)-\( ASPL \) or \( ASPL \) of the embedded topology. That is, RC is to trade increased space usage with decreased time. We use RC as baseline and optimize the mapping performance by using different mapping algorithms to investigate better time-space tradeoffs when imposing diverse limitations on embedding \( d \)-\( ASPL \)s or \( ASPL \)-\( s \).

As described in Sect. 3.2, for an embedded guest topology, given the required number of compute nodes \( n \) and the network degree \( d \), the Moore Bound of \( d \)-\( ASPL \) \( K_{n,d} \) and the lower bound of \( ASPL \) \( L_{n,d} \) are calculated according to the Eqs. (1) and (2) respectively. Since in this work our target is to compare the application mapping performances of different algorithms, we use a simple method, i.e., FCFS (First Come First Serve) as the job queuing policy, instead of other complicated backfilling strategies.

4.2 Workload Traces

To evaluate the performance of application mapping and job scheduling, we use a compound workload of NPB applications [17] and several real-world HPC workload traces [32].

4.2.1 NPB Applications

We use an event-driven simulator SimGrid [33] to simulate the executions of NPB applications [17], which include Block Tri (BT), Conjugate Gradient (CG), Fast Fourier (FT), Integer Sort (IS), Multi-Grid (MG) and Scalar Penta
Poisson timings for a approach to model parallel “rigid” jobs [1]. We generate number of resources during runtime by applying a common tions is from 5ms to 5s.

We assume that each application occupies a fixed number of resources during runtime by applying a common approach to model parallel “rigid” jobs [1]. We generate $a = [100, 1000]$ jobs as a workload with random arrival timings for a Poisson process with $\lambda = 200$. A workload is composed of the above NPB applications, and each job specifies the number (4, 16, 64 or 256) of required compute nodes. We use Class A as the problem size of FT, Class B as the problem size of BT, CG, LU, MG and SP.

4.2.2 PWA Workload Traces

We also include real supercomputer workload traces of user jobs by using the parallel workloads archive (PWA) datasets [32]. The datasets are portable and easy to parse since the same format is used for all the models and logs. Each archive in the PWA datasets uses a standard workload format [35] which concerns the arrivals of jobs and their basic resource requirements, namely the number of compute nodes and the processing time. In the evaluation, we exemplarily use four of the PWA datasets: NASA-iPSC, SDSC Par, SDSC SP2 and HPC2N, where the requested number of compute nodes are all less than our host network size. More detailed information about the PWA datasets can be found in [36].

For each trace log file, we mainly use the data fields of Number of Allocated Processors, Submit Time and Run Time to model application mapping in the evaluation. Because the real times found in the trace log files are exceedingly large for our simulation, for realistic simulation time and fair comparison, we use the first 10,000 jobs in each work- load log and divide both the Submit Time and Run Time by 1,000. The distribution of job sizes in a trace log file can be represented by its cumulative distribution function (CDF), which is defined as the probability that a specific sample is smaller than or equal to a given value. As observed in Fig. 8, the job sizes in each workload log are quite different but their distributions are approximately log-uniform.

4.3 Simulation Results

4.3.1 Graph Embedding

We take the metric of average graph embedding diameter as our evaluation objective, which represents the average shortest communication path hops of the node pairs during topology mapping.

The evaluation is divided into two groups based on the degree of the host network topology. The first group covers the degree-6 host topologies, including 3-D (16×8×8) torus and random (d6) topologies, while the second group covers the degree-10 host topologies, including 5-D (4×4×4×4×4) torus and random (d10) topologies. For fair comparison, all the host topologies are composed of 1,024 compute nodes.

When the host topologies are tori, our application mapping algorithm still prioritizes a non-random guest topology to have a low diameter and ASPL for small communication latency between compute nodes. Only if current non-random topology mapping does not exist, we consider to incrementally increase the maximum diameter or ASPL by using a diameter/ASPL-based mapping algorithm, e.g., D-MB-2 or L-V-1.

Due to space limitation, we only show the evaluation result by using L-V-1 as the topology mapping algorithm for different host topologies. Figure 9 shows the average diameter of topology embedding using different workloads over non-random and random host topologies. As shown in Fig. 9 (a), for NPB applications with different workload sizes, a random host topology brings a shorter average diameter than a torus host topology of identical degree (6 or 10). Also, for the same topology type (torus or random), a larger degree brings a shorter average diameter. Interestingly, a 5-D torus (degree-10) host topology takes certain advantage over a degree-6 random (d6) host topology in terms of average diameter. A similar tendency can be seen from the case of PWA traces, as shown in Fig. 9 (b). Notice that, in our topology mapping algorithms, all the embedded guest topologies are contiguous or connected to make their average diameter effective, while a non-contiguous topology brings a disconnected graph embedding which makes its diameter meaningless.

4.3.2 Job Scheduling

(1) Objective Functions

We introduce some notations and definitions to our evaluation of job scheduling. Assume that job $j$ out of all submitted jobs $J$ is scheduled on $n_j$ nodes out of a system of size $N$. The arrival time of job $j$ is denoted by $a_j$ and the completion time of job $j$ is denoted by $c_j$. The execution time of
job $j$ is denoted by $t_j$. The weight of job $j$ is defined as $n_j t_j$. The turnaround time of job $j$ is $c_j - a_j$, which 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 generate our evaluation functions by combining several simple and commonly used scheduling objectives. In this section, we exemplarily use four of the scheduling objectives as follows.

- **Makespan ($M$)**: the period from the first job arrival to the last job completion
  \[ M = \max_{j \in J} c_j - \min_{j \in J} a_j \]  
  \(3\)

- **Average Turnaround Time ($T$)**: the average response time from submission to completion of all submitted jobs
  \[ T = \frac{\sum_{j \in J} n_j (c_j - a_j)}{\sum_{j \in J} n_j t_j} \]  
  \(4\)

- **Average Slowdown ($S$)**: the average ratio of turnaround time to execution time of all submitted jobs
  \[ S = \frac{\sum_{j \in J} n_j (c_j - a_j)}{\sum_{j \in J} n_j t_j} \]  
  \(5\)

- **Overall Utilization ($U$)**: the overall system utilization
  \[ U = \frac{\sum_{j \in J} n_j t_j}{N (\max_{j \in J} c_j - \min_{j \in J} (c_j - t_j))} \]  
  \(6\)

(2) Non-random Topologies vs. Random Topologies

The settings of the host topologies are the same as those in Sect. 4.3.1, which include the degree-6 (3-D torus and random) and degree-10 (5-D torus and random) topologies. Figures 10 and 11 show the average turnaround time over different-degree host topologies. For NPB applications, as the workload size grows, the average turnaround time increases for any topology mapping algorithm, i.e., RC, D-MB-2 and L-V-1. Notice that, for the proposed application mapping algorithms, we also evaluated other diameter-based algorithms such as D-MB and D-MB-1 and other strict ASPL-based algorithms such as L-S-0.5. Their evaluation results are similar or slightly inferior to D-MB-2 and L-V-1 respectively, thus we omit their descriptions and show the evaluation results for only D-MB-2 and L-V-1.

Overall, a random topology outperforms a torus topology of identical degree in terms of average turnaround time. Furthermore, for the degree-6 topologies, as shown in Fig. 10(a), the advantage becomes larger as the workload size grows. However, this tendency does not exist for the larger-degree topologies, i.e., degree-10 topologies, as shown in Fig. 11(a). When the number of jobs is 1,000, compared to 3-D torus, a degree-6 random topology (d6)
reduces the average turnaround time by 39.3% using RC, by 66.1% using D-MB-2 and by 72.1% using L-V-1.

On the other hand, the diameter/ASPL-based topology mapping algorithms, i.e., D-MB-2 and L-V-1, have less average turnaround time than the baseline RC method. Comparatively, L-V-1 keeps a little advantage over D-MB-2 in all the cases. A similar tendency can be seen from the case of PWA traces, as shown in Fig. 10 (b) and 11 (b), where a random topology outperforms a torus topology of identical degree in terms of average turnaround time.

To summarize, random host topologies provide better job scheduling performance than non-random host topologies such as tori of identical degree in terms of average turnaround time. The advantage becomes more significant when the network size is large while the network degree is not large.

(3) Mapping Algorithms over Random Topologies

The host network is assumed to be a 1,024-node ($N = 1,024$) fully random topology with the degree of 6. Figure 12 shows the job scheduling performance for NPB applications. Figure 12 (a) depicts the relative makespan of diverse application mapping algorithms to the baseline RC method. Regardless of workload, D-MB performs better than RC due to its rigid diameter limitation on the Moore Bound during topology mapping, which offers shorter
Fig. 12 (a) shows the average makespan against the number of jobs, which reflects the average processing time of all the jobs from submission to completion. The baseline RC method presents the worst performance among all the topology mapping algorithms, because it causes more cases that an application is mapped on a large-diameter/ASPL guest topology as long as it is connected. In this case, the application execution time may increase, and even worse this also prevents the dispatch of its following applications due to node fragmentation and thus largely increases their wait times in the job queue.

Figure 12 (c) shows the comparison of performance on average slowdown, which reflects the average ratio of turnaround time to execution time. Thus, a large value indicates the heavier influence of wait time. Comparatively, D-MB performs better than RC in terms of average turnaround time and average slowdown, because its topology mapping diameter is strictly limited to the Moore Bound. However, the strict limitation on the Moore Bound also prevents the further improvement of its topology mapping performance when compared to D-MB-1 and D-MB-2, which relax the topology mapping diameter to slightly larger values and present not much difference. The L-S-0.5 and L-V-1 algorithms use ASPL instead of diameter as the topology mapping limitation and also present comparable performance to D-MB-1 and D-MB-2. As the workload becomes heavy, e.g., the number of jobs is 1,000, L-V-1 shows the best performance among all the topology mapping algorithms.

A similar tendency can be seen from Fig. 12 (d), which reflects the comparison of node utilization on the system over the makespan. Overall, the ASPL-based algorithms (L-S-0.5 and L-V-1) show a little advantage over the diameter-based algorithms (D-MB, D-MB-1 and D-MB-2). Especially, when the workload is heavy, e.g., the number of jobs is 1,000, L-V-1 still presents the best performance, and reduces the makespan up to 48.0% and the average turnaround time up to 78.1% and improves up to 1.9x system utilization when compared to the baseline RC method.

For PWA workload traces, as shown in Fig. 13, in most cases the diameter/ASPL-based application mapping outperforms the baseline RC topology mapping in terms of job scheduling performance. However, when the requested compute nodes are small or the application execution times are obviously shorter than the job arriving intervals (like in SDSC SP2 and HPC2N), there is not much difference in average turnaround time and average slowdown among different topology mapping algorithms. In this case, most
compute nodes in the whole network keep available so that
an incoming job can be immediately dispatched using any
topology mapping algorithm. Another exception is the case
that D-MB performs worse than other topology mapping al-
gorithms in SDSC Par in terms of average turnaround time
and average slowdown. This is because the strict limita-
tion on the Moore Bound of topology mapping diameter
may harm the job scheduling performance due to large wait
time for a guest topology with rigid limitation on diam-
eter. A simple optimization is to relax the limitation to a
slightly larger value, e.g., the Moore Bound plus one or two.
The phenomenon that the performance of L-V-1 degrades
in terms of makespan under HPC2N is due to the largely inten-
sive job arrival within a compact time interval before com-
pletion. In this case, the incrementally increased embedding
ASPL upper bounds (e.g., 75% over lower bound) may delay
the completion time of all dispatched jobs, which however
do not degrade their average turnaround time. Overall, the
ASPL-based L-V-1 algorithm is recommended in any case
due to its stable improved job scheduling performance.

5. Conclusion

Accompanied by significant technology progress based on
big-data processing, recent emerging applications present
irregular or unpredictable communication patterns. In this
context, for the purpose of improving the performance of
job scheduling, we targeted the applications with unknown
or uncertain communication patterns to the system, and ex-
plored their topology mapping algorithms. We proposed us-
ing random network topologies for such applications, and
explored the diameter/ASPL-based topology mapping stra-
egy to comparatively analyze its application mapping per-
formance over non-random and random network topologies.

By a series of event-driven simulations with a com-
pound application workload, we showed that, compared to
a non-random host topology such as torus, a random host
topology of identical degree favors the job scheduling per-
formance. The superiority becomes significant when the
network size is large while the network degree is not large.
When compared to non-random topologies, random topol-
ogies can reduce the average turnaround time up to 39.3% by
a random connected mapping method and up to 72.1% by a
diameter/ASPL-based mapping algorithm. We also showed
that, the proposed diameter/ASPL-based application map-
ing strategy further improves the job scheduling perfor-
ance when compared to the baseline random connected
(RC) topology mapping method. Overall, the ASPL-based
L-V-1 algorithm is recommended for topology mapping
over random network topologies by reducing the makespan
up to 48.0% and the average turnaround time up to 78.1%
and improving up to 1.9x system utilization.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Num-
ber 19H01106.

References

[1] D.G. Feitelson, Workload Modeling for Computer Systems Perform-
ance Evaluation, Cambridge University Press, March 2015.
[2] O. Tuncer, V.J. Leung, and A.K. Coskun, “Pacman: Topology map-
ing of unstructured communication patterns onto non-contiguous
allocations,” Proc. 29th ACM on International Conference on Su-
percomputing, pp.37–46, 2015.
[3] 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.
[4] T. Hoefler and M. Snir, “Generic topology mapping strategies for
large-scale parallel architectures,” Proc. International Conference
on Supercomputing, ser. ICS’11, pp.75–84, New York, NY, USA,
ACM, 2011. [Online]. Available: http://doi.acm.org/10.1145/
1995896.1995909
[5] [Online]. Available: https://en.wikipedia.org/wiki/Cray_XT6
[6] [Online]. Available: https://ja.wikipedia.org/wiki/Blue_Gene
[7] J. Galvez, N. Jain, and L. Kale, “Automatic topology mapping of di-
verse large-scale parallel applications,” ICS 2017, ser. Proc. Interna-
tional Conference on Supercomputing, Association for Computing
Machinery, June 2017.
[8] A. Singla, C.-Y. Hong, L. Popa, and P.B. Godfrey, “Jellyfish: Net-
working Data Centers Randomly,” Proc. USENIX Symposium on
Network Design and Implementation (NSDI), pp.225–238, 2012.
[9] M. Koibuchi, H. Matsutani, H. Amano, D.F. Hsu, and H. Casanova,
“A Case for Random Shortcut Topologies for HPC Interconnects,”
Proc. International Symposium on Computer Architecture (ISCA),
pp.177–188, 2012.
[10] M. Koibuchi, I. Fujiwara, H. Matsutani, and H. Casanova, “Layout-
conscious random topologies for hpc off-chip interconnects,” 19th
International Conference on High-Performance Computer Architectu-
re (HPCA), pp.484–495, Feb. 2013.
[11] F. Chaix, I. Fujiwara, and M. Koibuchi, “Suitability of the random
topology for hpc applications,” 2016 24th Euromicro International
Conference on Parallel, Distributed, and Network-Based Processing
(PDP), pp.301–304, Feb. 2016.
[12] A. Bhatele, T. Gamblin, S.H. Langer, P. Bremer, E.W. Draeger, B.
Hamann, K.E. Isaac, A.G. Landge, J.A. Levine, V. Pascucci, M.
Schulz, and C.H. Stilt, “Mapping applications with collectives over
sub-communicators on torus networks,” SC ’12: Proc. International
Conference on High Performance Computing, Networking, Storage
and Analysis, pp.1–11, Nov. 2012.
[13] M. Deveci, S. Rajamanickam, V.J. Leung, K. Pedretti, S.L. Olivier,
D.P. Bunde, U.V. Catalyurek, and K. Devine, “Exploiting geometi-
ric partitioning in task mapping for parallel computers,” 2014 IEEE
28th International Parallel and Distributed Processing Symposium,
pp.27–36, May 2014.
[14] H. Subramoni, S. Potluri, K. Kandalla, B. Barth, J. Vienne, J.
Kesler, K. Tomko, K. Schulz, A. Moody, and D.K. Panda, “De-
sign of a scalable infiniband topology service to enable network-
topology-aware placement of processes,” SC ’12: Proc. International
Conference on High Performance Computing, Networking, Storage
and Analysis, pp.1–12, Nov. 2012.
[15] A. Gupta, L.V. Kalé, D. Milojicic, P. Faraboschi, and S.M. Balle,
“Hpc-aware vm placement in infrastructure clouds,” 2013 IEEE
International Conference on Cloud Engineering (IC2E), pp.11–20,
March 2013.
[16] Y. Hu, “Topology mapping of parallel applications onto random
allocations,” The 21st International Conferences on High Perform-
ance Computing and Communications (HPCC-2019), Zhangjiajie,
China, pp.1437–1444, Aug. 2019.
[17] The NAS Parallel Benchmarks, http://www.nas.nasa.gov/Software/
NPB/.
[18] Y. Hu and M. Koibuchi, “Diameter/aspl-based mapping of applica-
tions with uncertain communication over random interconnection
networks,” Proc. IEEE International Conference on Parallel And Distributed Systems (ICPADS), ser. ICPADS ’19, pp.249–258, Piscataway, NJ, USA, IEEE Press, Dec. 2019.

[19] H. Cai, V.W. Zheng, and K.C. Chang, “A comprehensive survey of graph embedding: Problems, techniques and applications,” CoRR, vol.abs/1709.07604, 2017. [Online]. Available: http://arxiv.org/abs/1709.07604

[20] S.A. Smith, C.E. Cromey, D.K. Lowenthal, J. Domke, N. Jain, J.J. Thiggarajan, and A. Bhatfe, “Mitigating inter-job interference using adaptive flow-aware routing,” SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, pp.346–360, Nov. 2018.

[21] [Online]. Available: https://bluewaters.ncsa.illinois.edu/torus-viewer

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

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

[24] 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

[25] 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,” 2014 IEEE International Conference on Cluster Computing (CLUSTER), pp.140–148, Sept. 2014.

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

[27] 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

[28] T. Agarwal, A. Sharma, A. Laxmikant, and L.V. Kale, “Topology-aware task mapping for reducing communication contention on large parallel machines,” Proc. 20th IEEE International Parallel Distributed Processing Symposium, pp.10, April 2006.

[29] D.K. Dror, G. Feitelson, and D. Tsafrir, “Experience with using the Parallel Workloads Archive,” Journal of Parallel and Distributed Computing, vol.74, no.10, pp.2967–2982, Oct. 2014.

[30] S.J. Chapin, W. Corne, D.G. Feitelson, J.P. Jones, S.T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, “Benchmarks and Standards for the Evaluation of Parallel Job Schedulers,” Job Scheduling Strategies for Parallel Processing, vol.1659, pp.66–89, 1999.

[31] 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

[32] 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

[33] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, “Versatile, scalable, and accurate simulation of distributed applications and platforms,” Journal of Parallel and Distributed Computing, vol.74, no.10, pp.2899–2917, June 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[34] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoolhi, 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/