Mapping Matters: Application Process Mapping on 3-D Processor Topologies

Jonas H. Müller Korndörfer[1], Mario Bielert[2], Laércio L. Pilla[3], and Florina M. Ciorba[1]

[1] Department of Mathematics and Computer Science, University of Basel, Basel, Switzerland, Email: jonas.korndorfer@unibas.ch and florina.ciorba@unibas.ch
[2] Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden, Dresden, Germany, Email: mario.bielert@tu-dresden.de
[3] Laboratoire de Recherche en Informatique, CNRS & Univ. Paris-Saclay, Orsay, France, Email: pilla@lri.fr

June 4, 2020

Abstract

Applications' performance is influenced by the mapping of processes to computing nodes, the frequency and volume of exchanges among processing elements, the network capacity, and the routing protocol. A poor mapping of application processes degrades performance and wastes resources. Process mapping is frequently ignored as an explicit optimization step since the system typically offers a default mapping, users may lack awareness of their applications' communication behavior, and the opportunities for improving performance through mapping are often unclear. This work studies the impact of application process mapping on several processor topologies. We propose a workflow that renders mapping as an explicit optimization step for parallel applications. We apply the workflow to a set of four applications, twelve mapping algorithms, and three direct network topologies. We assess the mappings' quality in terms of volume, frequency, and distance of exchanges using metrics such as dilation (measured in hop·Byte). With a parallel trace-based simulator, we predict the applications' execution on the three topologies using the twelve mappings. We evaluate the impact of process mapping on the applications' simulated performance in terms of execution and communication times and identify the mappings that achieve the highest performance in

both cases. To ensure the correctness of the simulations, we compare the pre- and post-simulation results. This work emphasizes the importance of process mapping as an explicit optimization step and offers a solution for parallel applications to exploit the full potential of the allocated resources on a given system.

1 Introduction

The growing amount of computing elements in HPC systems inherently presents a new bottleneck in terms of the necessary communication for data distribution as well as controlling the application processing elements, e.g., tasks, threads, and processes. Therefore, the performance of parallel applications highly depends on their communication behavior.

Nowadays, parallel applications execute on a broad range of parallel computing architectures, from large supercomputers to embedded low-power architectures. When these applications execute on parallel systems, their communication time is affected by how intensely their processing elements exchange data, by the capacity and performance of the network links, and by *the placement of processing elements on the computing resources*.

Application placement is typically the result of a mapping algorithm. Efficient application placement on modern hardware architectures is of paramount importance for performance [12, 24, 29, 39, 48]. A poor choice of the mapping algorithm may lead to larger communication latencies and, therefore, to significant performance loss and energy waste. A plethora of *communication and/or topology-aware mapping* algorithms emerged over the years in the literature to improve application process placement (see [29] for a recent overview).

Process mapping, also known as topology mapping or application process placement, is an active area of research with a vast history that includes algorithms whose performance benefits have been recorded in numerous situations. For instance, communication and/or topology-aware mapping techniques combine information about the target application (its *communication pattern* or *virtual topology*) and the target system (its *physical topology*) to take mapping decisions following a performance objective, e.g., minimizing *dilation*, *distance*, *volume·distance*, or congestion [29]. Other approaches simply employ *space-filling curves* (SFCs) [13] which essentially generate mappings based on common communication patterns.

Given the need to achieve high performance and mitigate resource waste, many application developers and users face the following questions:

1. *How can one verify if an application is suffering from poor communication performance?*
2. *How and how much does mapping impact an application's performance on a given system?*
3. *Which mapping algorithm is the highest performing for a given application–system pair?*

The absence of simple answers to these questions has severe consequences. While a naïve process mapping may lead to performance loss, an inapt mapping may cause longer execution times in addition to the overhead associated with generating the mapping itself. Over time, such performance loss translates into congested or wasted resources and increased energy consumption. Moreover, performing repeated experiments to identify the highest performing mapping algorithm is neither sustainable nor scalable. This results in process mapping often being ignored as an explicit application optimization step.

In this work, we study the impact of application process mapping on several processor topologies. We propose a workflow to support mapping as an explicit application optimization step. We apply the workflow to four applications, mapped using twelve mapping algorithms, executing on three direct network topologies.

This work makes the following contributions: (i) it proposes a generic workflow to support process mapping as an explicit application optimization step; (ii) it provides an analysis on the predicted mapping benefit for a given application–system pair; and (iii) it contributes a Python-based library with well-known topology mapping algorithms from the literature.

This work is organized as follows. The work related to application optimization through careful placement is reviewed in Section 2. The generic workflow for mapping applications onto processor topologies is introduced and described in Section 3. The application characteristics and evaluation metrics are presented in Section 4, while the processor topologies and network models are discussed in Section 5. Section 6 provides a description of the mapping algorithms. The design of performance experiments and their evaluation and analysis are presented in Section 7. Section 8 concludes the work and outlines future work directions.

2 Related Work

Process placement on modern hardware architectures has been studied from various dimensions and in various contexts, leading to a significant body of work emerging in the literature over the years. Recent surveys provide an overview of existing solutions and open problems on the topic [29] [39] [12] [45].

The approach taken in this work involves: parallel MPI applications, algorithmic strategies for topology mapping, three-dimensional direct network topologies, communication models, application tracing, and trace-driven simulation.

Most existing work considers parallel MPI applications when studying process mapping. MPI point-to-point calls have been found in a recent study [38] to be more prominently used than either persistent point-to-point or one-sided MPI calls. In this work, we study the effect of process mapping on the performance of MPI point-to-point calls.

Hoefler et al. [29] classify the algorithmic strategies for topology mapping into four categories. The algorithms considered in this work fall into three of these categories: *greedy* includes the `Peano`, `Hilbert`, `Gray`, `sweep`, and `scan`

SFCs, `greedy`, `FGgreedy`, `greedyALLC`, and `topo-aware`, *graph partitioning* includes `bipartition` and `PaCMap`, while `Bokhari` is an *isomorphism-based* algorithm. We do not include any *subgraph isomorphism-based* algorithms, because we only consider bijective mappings in this work.

Process mapping is influenced by the underlying network topology. Therefore, mapping has been studied in the context of modern network topologies (`torus`, `fat-tree`, and `Dragonfly`) and technologies (InfiniBand, Ethernet, BlueGene, Cray, and others) [29]. LibTopoMap [30] is a generic library of graph mapping heuristics (recursive bisection, k-way partitioning, simple greedy strategies, and Cuthill-McKee). Mapping with LibTopoMap is based on similarity metrics (e.g., bandwidth of the adjacency matrices), employs rank reordering for MPI applications, and can be used on various network topologies and technologies. While the library is generic and versatile, it is not directly usable in a simulated environment where certain practical effects in the software stack are abstracted and a study of process mapping can concentrate on specific aspects such as the communication cost. Simulation is also important for the co-design of applications and future systems and is the approach we take in the present work.

MPIPP [19] is a framework dedicated to MPI applications with arbitrary virtual topologies executing on SMP clusters and multi-clusters. Similar to our work, the framework employs application tracing to obtain the communication behavior of the application. In contrast to this work, the communication pattern is stored as a communication graph which is placed on the topology graph (determined on-the-fly via a parallel ping-pong mechanism). Also different from our work, they only consider graph partitioning algorithms and conduct direct experiments on SMP clusters.

Rodrigues et al. [46] use a purely quantitative approach to apply resource binding for MPI applications and reduce communication costs in multicore nodes. Mercier and Clet-Ortega [40] use a similar but qualitative approach to the same problem. Both works [46] [40] employ graph partitioning to compute the mapping.

Automated mapping has also been studied in the context of regular application communication graphs on 2-D and 3-D `mesh` and `torus` networks [15]. There, the virtual topology is also represented as a graph and the mapping heuristics are chosen to optimize for hop·Byte. In our work, the virtual topology is represented as an adjacency matrix, while the mapping heuristics also optimize for dilation (measured as hop·Byte) on 3-D `mesh` and `torus` topologies, complemented by the 3-D `HAEC Box` topology.

EagerMap [20] employs a greedy topology mapping algorithm for hierarchical machine topologies (trees). Its algorithm groups together the application's processes that show the highest affinity based on the communication matrix. Although the algorithm has been adapted to handle arbitrary network topologies, it requires hierarchical multicore nodes. In contrast, in this work we concentrate on 3-D network topologies.

Rico-Gallego et al. [45] surveyed prominent communication performance models in high-performance computing, which are often tested in simulation.

They state that future models will need to take into account accurate performance and energy modeling. In this work, we employ a contention-oblivious communication model called *network coding dynamic resilient* (NCD_r) [43] that transmits messages efficiently, reliably, and with minimal energy costs. NCD_r is implemented in HAEC-SIM [16], the trace-based simulator used in this work.

In addition to careful process mapping, Sensi et al. [48] show that application-aware routing outperforms application-agnostic routing. Similarly, in this work we employ *shortest path routing* and argue that, in addition, communication-aware mapping is needed and show that, in certain cases, it outperforms communication-oblivious mapping.

Trace-driven simulation has also recently been used by Tsuji et al. [49] to study application behavior on future systems. Their workflow is very similar to ours with the difference that their focus is to support the analysis of applications larger than the real MPI traces from existing systems and they do not explicitly consider process mapping. While we are also concerned with scalability (planned for future work), in the present work we concentrate on answering the three research questions (Section 1) for modern application-system pairs.

Kenny et al. [34] consider the influence of the network and its parameters on the performance of parallel MPI applications. They decompose the application time spent in MPI into: communication, synchronization, and software stack components. Through the combination of Bayesian inference and trace replay, they found that synchronization and MPI software stack overheads are at least as important as the network itself in determining time spent in communication routines. In this work, we explicitly study the impact of network topology and its parameters under various process mapping strategies. Measuring the time spent in synchronization and in various components of the software stack is highly complex and will be incorporated into our workflow in the future.

3 Proposed Workflow

This work proposes a workflow, illustrated in Fig. 1, to render mapping as an explicit application optimization step. Following the Y-chart design methodology [35], the workflow provides application optimizers quantitative data obtained by analyzing the performance of applications on topologies for a given set of mapping algorithms.

The steps in red (Section 4) refer to the extraction and analysis of performance metrics from the application. These steps are independent from the mapping strategies or target topologies. The topology information (in orange, Section 5) includes the specification of the target topology, communication model, and path selection. The blue workflow steps (Section 6) combine the information gathered from applications and topologies to generate various process mappings with mapping algorithms from the literature.

The green steps (Section 7) propose the evaluation and analysis of the outputs from previous steps assessing the performance gains of various mappings for a given application–system pair. We evaluate and analyze performance in

two phases. The first phase corresponds to analysis *pre-simulation* of the mapping quality using metrics that do not require direct nor simulated experiments. The second phase corresponds to analysis *post-simulation*, to quantify the application performance gains predicted via simulation. Finally, we compare the *pre-simulation* results in terms of communication volume, frequency, and distance against those extracted *post-simulation*, to ensure the correctness of the simulation and to assess the usefulness of the pre-simulation metrics.

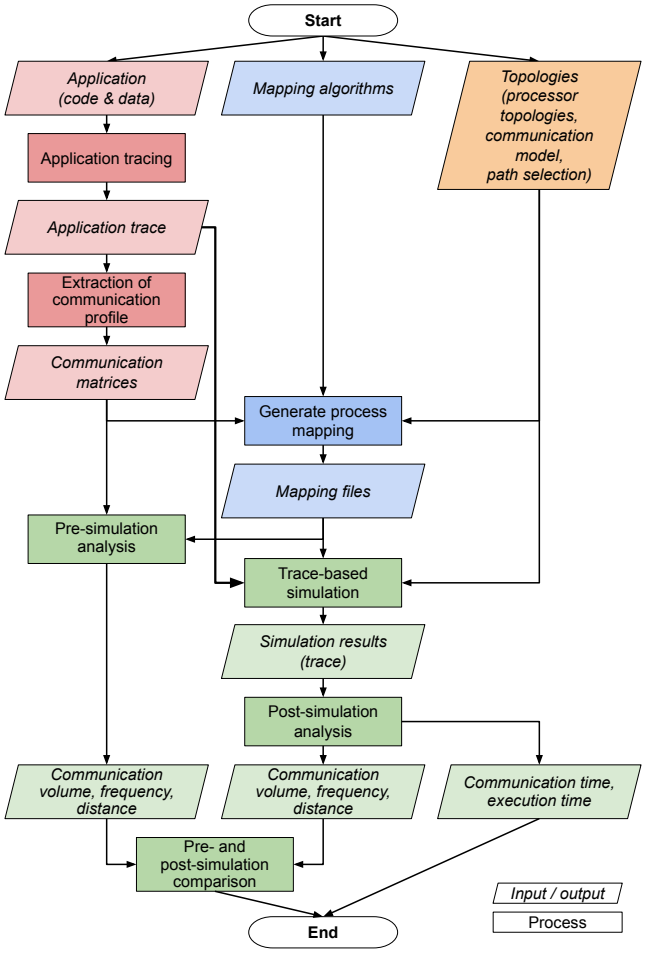


Figure 1: Rectangles represent actions, while parallelograms represent information that actions need as input or produce as output. The shape colors represent different types of workflow steps: red steps relate to applications; blue steps denote mapping-related activities; the orange step concerns machine topologies; and green steps indicate simulation, performance evaluation, and analysis.

4 Parallel Communication-Intensive Applications

4.1 Applications and Tracing

We consider four target applications from well-known benchmark suites. From the NAS parallel benchmarks [14] [10], we investigated CG and BT-MZ, input size class C [1]. NAS CG computes an approximation of the smallest eigenvalue for a large, sparse, and symmetric matrix. It stresses several irregular long distance communications. NAS BT-MZ is a newer version of the Block Tri-diagonal solver (BT) which was re-designed to exploit multiple levels of parallelism (MPI and OpenMP). From the CORAL2 [9] benchmark suite, we selected AMG [2] with the default input (problem 1). AMG is a multigrid solver that generates many small messages and stresses memory and network latency. From the CORAL [8] benchmark suite, we used LULESH [3, 32] and set the time iterations count to 1,000. LULESH is a proxy application that simulates a variety of problems that describe the motion of materials relative to each other when subject to forces.

Given the application source codes and input data, we executed and traced them on a parallel system with Intel Broadwell E5-2640 v4 [5] processors organized into 2 sockets, each with 10 CPU cores. The nodes in the system are connected by an Intel Omni-Path network with 100 Gbit/s and a two-level `fat-tree` interconnection topology. The applications executed on 16 nodes of this system, using 4 MPI ranks per node (2 on each socket of the node) for a total of 64 MPI ranks. To record the application trace, we used the Score-P v. 4.1 [36] measurement infrastructure.

Table 1 summarizes for each application the time spent in computation versus communication as recorded in the trace. The computation time includes all non-MPI functions. The MPI total communication time includes time spent in MPI point-to-point operations as well as other MPI calls (MPI other).

Table 1: Computation and communication times and percentages for the selected applications.

	NAS CG		**NAS BT-MZ**		**CORAL2 AMG**		**CORAL LULESH**	
Computation total	140.45 s	2.8 %	860.88 s	84.4 %	711.32 s	75.8 %	14 231.36 s	83.2 %
MPI_Send	3628.63 s	71.3 %	–	–	0.17 s	0.0 %	–	–
MPI_Receive	–	–	–	–	1.64 s	0.2 %	–	–
MPI_Isend	–	–	4.06 s	0.4 %	3.13 s	0.3 %	19.74 s	0.1 %
MPI_Irecv	1.71 s	0.0 %	0.53 s	0.1 %	0.41 s	0.0 %	2.29 s	0.0 %
MPI_Wait	1301.72 s	25.6 %	–	–	–	–	729.47 s	4.3 %
MPI_Waitall	–	–	126.97 s	12.4 %	90.39 s	9.6 %	4.04 s	0.0 %
MPI other	13.78 s	0.3 %	27.91 s	2.7 %	130.88 s	13.9 %	2114.95 s	12.4 %
MPI total	4945.84 s	97.2 %	159.47 s	15.6 %	226.62 s	24.0 %	2870.59 s	16.8 %

4.2 Process-logical Communication Matrices

The next step involves extracting the application communication behavior from the trace into a *process-logical communication matrix*. The process-logical com-

munication matrix of the applications can also be collected via other approaches in a comma-separated-value (CSV) format. The extracted information can represent various quantities such as the number of message exchanges, the volume of exchanges, the average message transfer time, and others.

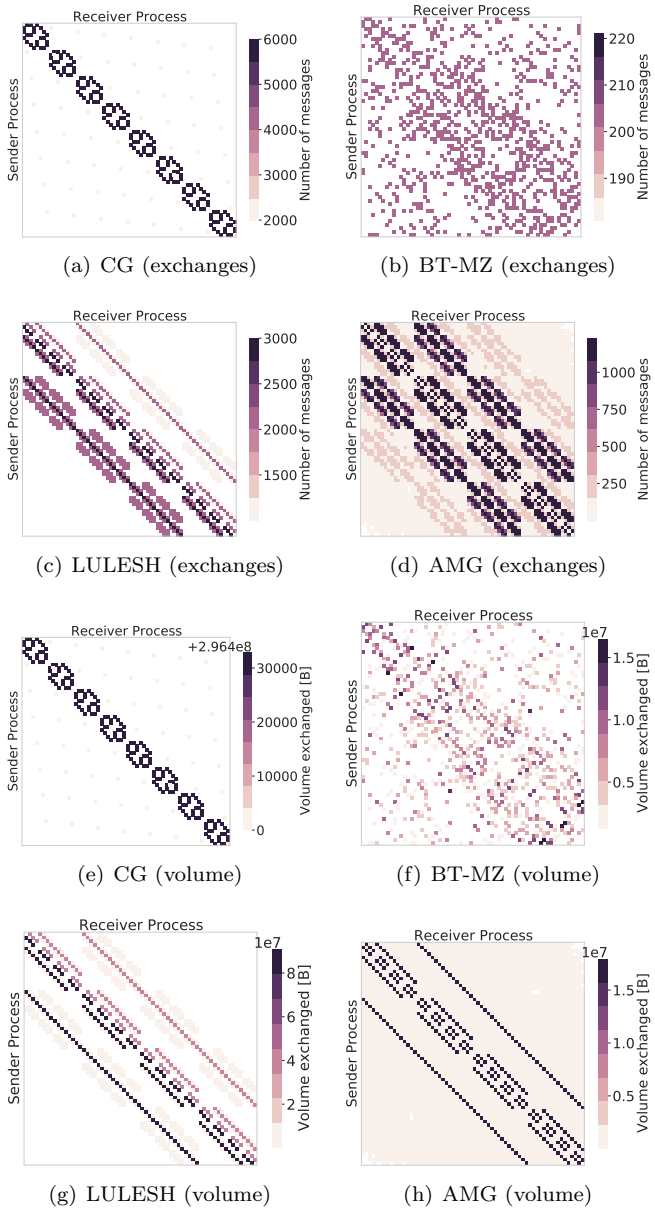
Fig. 2 illustrates the two types of process-logical communication matrices employed in this work: number of point-to-point message exchanges (*commMatrix count*) and volume (in Byte) of point-to-point message exchanges (*commMatrix size*). BT-MZ, LULESH, and AMG exhibit highly irregular communication patterns which was an important criterion in our applications' selection and already indicates potential room for performance improvement via explicit mapping.

4.3 Application-related Communication Metrics

Process-logical communication matrices provide a first view into an application's communication behavior. The structure of a communication matrix alone may prove insufficient to predict if and how much will an application benefit from careful process mapping [18]. To enable such predictions, *communication metrics* (or *matrix-based statistics*) have previously been proposed and tested in the context of thread mapping on shared-memory machines [22, 23] and process mapping on hierarchical machines (multicore nodes in `fat-tree` topologies) [18]. Such metrics are useful when we are interested in predicting performance gains through mapping, but also to triage applications for which mapping studies can not be performed (e.g., when multiple applications need to be tested while access to resources is very limited) or for which mapping is not expected to yield performance improvements. *It is also important to emphasize that this work is the first to employ these communication metrics in 3-D topologies.*

The following communication metrics are considered in this work: *Communication Amount* (*CA*) indicates the average inter-process communication [23]; *Communication Balance* (*CB*) measures the divergence of the most communicating process from the others [22]; *Communication Centrality* (*CC*) quantifies the dispersion of communication from the main diagonal in the communication matrix [18]; *Communication Heterogeneity* (*CH*) denotes the average communication variance of each process [23]; *Neighbor Communication Fraction* (*NBC*) represents the fraction of communication between processes with close rank identifiers (nearest neighbors) [18]; and *Split Fraction SP(k)* measures the fraction of communication among processes in k^2 blocks [18].

Depending on the communication matrix used as input (*commMatrix count* or *commMatrix size*) the above metrics represent different values. For example, *CA* calculated from *commMatrix count* denotes the average number of messages exchanged between all processes, while *CA* calculated from *commMatrix size* indicates the average volume (in Byte) exchanged between all processes. *For all metrics, higher values indicate a higher potential benefit via careful process mapping.*



(a) CG (exchanges) (b) BT-MZ (exchanges)

(c) LULESH (exchanges) (d) AMG (exchanges)

(e) CG (volume) (f) BT-MZ (volume)

(g) LULESH (volume) (h) AMG (volume)

Figure 2: Applications' communication behavior for executions with 64 MPI processes. The top row shows the communication matrices in terms of point-to-point messages (heat bar: count), while the bottom row represents them in terms of volume (heat bar: Byte). The X and Y axes represent the receivers and senders, respectively. A white cell denotes an absence of communication exchanges. The cell in the upper left corner represents exchanges from and to process 0.

We computed the values of the six above metrics for the four applications using the formulae [7] proposed in [18] except for *CA*, which is not mentioned therein and, thus, we obtained it independently in this work. We computed the Split Fraction of applications for $k = 4$ and $k = 16$ as they represent a portion and a full plane in the 3-D topologies considered, respectively (more details in Section 5).

Tables 2 and 3 present the values of these metrics based on the *commMatrix count* and *commMatrix size* of the four applications, respectively. The first row of values presents the sum ($\sum$) of the values of all cells in each communication matrix. All metric values are rounded upward to three decimal places. **Bold** values indicate the largest value for each metric.

Table 2: Communication metrics extracted from *commMatrix count*.

	CG	**BT-MZ**	**AMG**	**LULESH**
$\sum$	1 279 232	182 910	1 257 257	**1 692 936**
CA	312.313	44.656	306.948	**413.314**
CB	0.000	0.289	**0.418**	0.413
CC	0.061	0.339	**0.369**	0.325
CH	0.046	**0.171**	0.119	0.073
NBC	0.700	**0.963**	0.910	0.858
SP(4)	0.387	**0.941**	0.898	0.858
SP(16)	0.074	**0.651**	0.637	0.589

Table 3: Communication metrics extracted from *commMatrix size*.

	CG	**BT-MZ**	**AMG**	**LULESH**
$\sum$	**75 884 703 744**	4 785 761 760	5 431 711 224	20 161 171 008
CA	**18 526 539.000**	1 168 398.867	1 326 101.373	4 922 160.809
CB	0.000	0.205	**0.273**	0.258
CC	0.061	**0.292**	0.163	0.157
CH	0.059	0.025	**0.063**	0.041
NBC	0.750	**0.954**	0.686	0.677
SP(4)	0.469	**0.926**	0.685	0.677
SP(16)	0.187	**0.584**	0.354	0.344

Several observations can be drawn from Tables 2 and 3. BT-MZ shows the largest values for most metrics, indicating that it has the largest potential to benefit from careful process mapping. AMG ranks second in terms of the number of highest values. Yet, one can note that this occurs for the *CB*, *CC*, and *CH* metrics. As these metrics are the ones that show the smallest ranges, they may not help in sufficiently differentiating the applications, which makes it difficult to estimate the potential impact of mapping AMG compared to the other applications.

The *CA* metric shows that CG exchanges the largest volumes and LULESH exchanges the highest message counts among all applications. These two observations indicate that they may be the most sensitive to changes in bandwidth and latency, respectively. *CB* obtained from both communication matrices are zero for CG. This means that all processes exchange exactly the same total volume and total number of messages.

These metrics allow us to make predictions of which applications are expected to benefit from careful mapping. However, the quantification of such benefits is strongly related to the specific network topology used. For instance, no changes would be observable across mappings on a fully connected topology. The next section describes the characteristics of network topologies.

5 Interconnection Network Topologies

One of the three inputs to the workflow illustrated in Fig. 1 is the target topology (in orange). The mapping problem might only be apparent when thinking of parallel applications running on different nodes of an HPC system. However, from an abstract perspective, a self-similar fractal-like observation unfolds:

Most modern networks can be seen as connected to an encompassing interconnection network or as an interconnection network to their sub-networks. For instance, an HPC system comprises several isles, a node contains multiple NUMA domains, and a SoC processor architecture contains several components. Each of the above systems employs a network topology containing another subnetwork. Hence, there are numerous opportunities to analyze the impact of different mapping strategies on the performance of applications executing in such systems.

5.1 Direct 3-D Topologies

Nowadays, HPC systems are usually built with sophisticated network topologies, e.g., `butterfly`, `fat-tree`, `Dragonfly`. Yet in practice, not every application receives an allocation that allows it to exclusively utilize the entire network. Instead, a resource allocation forms a virtual subnetwork, which has a simpler topology and requires local process mapping. Furthermore, other networks still employ simpler topologies. For instance, the Intel SkyLake SP architecture uses a 2-D `torus` topology to interconnect all cores on a chip [47].

Influenced by these observations, we consider topologies that arrange 64 nodes in a $4 \times 4 \times 4$ 3-D fashion as a proxy for more complex interconnection topologies. Such an arrangement already allows a rich selection of network topologies, namely 3-D `mesh`, 3-D `torus`, and the 3-D `HAEC Box` topology.

The `HAEC Box` topology stems out of the HAEC project [25] which envisions a novel highly adaptive and energy-efficient network topology, with nodes arranged on four vertically laid out (in the Z dimension) boards and each board contains 4×4 nodes (in the XY plane). Nodes of one board are interconnected

Table 4: Network link characteristics used in simulation.

Link type	Bandwidth	Latency	Bit error rate
Wireless	100 Gbit s^{-1}	100 ps	1e−8
Optical	250 Gbit s^{-1}	10 ps	1e−12

using fast optical links in a 2-D `torus`. A fully-connected wireless interconnection array facilitates inter-board communication (in the Z dimension).

We assume that all topologies considered in this work employ future-generation network links. Table 4 shows the characteristics of such future-generation links. While the `mesh` and `torus` topologies only contain optical links, the `HAEC Box` is a *heterogeneous* network topology and employs both wireless (across boards) and optical (on-board) links. The heterogeneity of the `HAEC Box` links mandates a systematic analysis of the suitability of mapping algorithms, such as the one presented in this work.

5.2 Path Selection

In the pre- and post-simulation analyses (Sections 7.1 and 7.3), we exclusively use static path selection, i.e., the path of individual packets through the network only depends on the source and destination positions and remains unchanged during execution. The 3-D `mesh` and 3-D `torus` topologies employ *shortest-path XYZ dimension order routing* (DOR) to predetermine the path that the messages will take. XYZ DOR first routes messages along the X dimension until the X coordinate of the current hop destination is equivalent to the X coordinate of the message destination. The hop selection is applied similarly along the Y and then the Z dimensions.

On the `HAEC Box` topology, the simulation uses the same XYZ DOR algorithm when the message source and destination are on the same board. When the message has to hop across boards, the first hop is to the node on the neighboring board with the same X and Y coordinates as the destination node. After the first hop across boards, every subsequent hop only has to follow along the Z dimension until the message reaches its destination.

5.3 Communication Model

To model communication, we use a contention-oblivious implementation of the NCD_r network model [44], implemented in HAEC-SIM [16]. This model describes the separation of message bodies into packets and the duration of efficient, reliable, minimal energy cost transmissions over the network based on future-generation wireless and optical links.

6 Mapping Applications onto Parallel Machines

This section describes the blue workflow steps (Fig. 1) and the mapping algorithms and clarifies their implementation and use in this work.

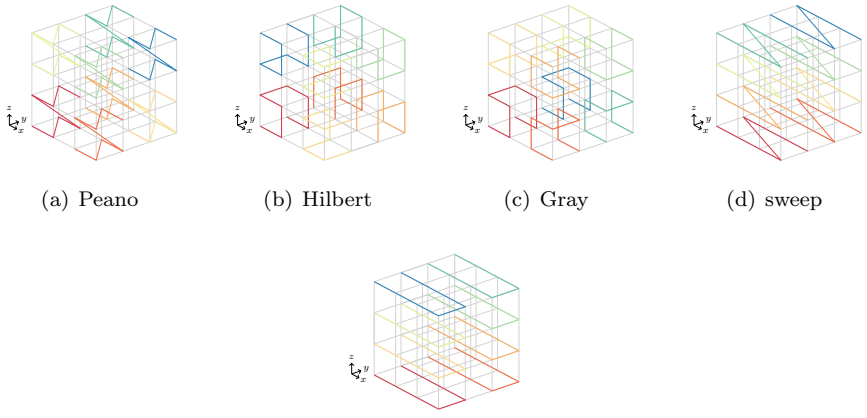
6.1 *MapLib*, a Library of Mapping Algorithms

We implemented twelve mapping algorithms from the literature (see Section 2) in a new Python library called *MapLib* [6]. The algorithms generate mappings for the three 3-D topologies considered herein: `mesh`, `torus`, and `HAEC Box`. The support for other topologies is considered as future work. Additional details regarding the installation and particular configurations are offered together with the library. The library generates ASCII mapping files, the structure of which is exemplified in [16].

MapLib implements: (i) Communication- and topology-oblivious mapping strategies (Section 6.2), which do not take into account the communication matrices of the applications nor the target processor topologies. These algorithms follow a predetermined node ordering to map all processes to the available nodes and produce deterministic mappings. (ii) Communication- and topology-aware mapping strategies (Section 6.3), which consider both communication matrices of the applications and target processor topologies. These algorithms produce different mappings for a given application–system pair.

6.2 Communication- and Topology-Oblivious Mapping

The communication- and topology-oblivious mapping strategies implemented are five space filling curves (SFCs), illustrated in Fig. 3, that are extensively used as mapping schemes. They map discrete multi-dimensional spaces onto one-dimensional spaces [41]. SFCs were discovered in the nineteen century by Peano [42], followed by Hilbert [28]. Numerous SFC variations have been studied since then.



(a) Peano (b) Hilbert (c) Gray (d) sweep

(e) scan

Figure 3: Illustration of the five SFCs on a 3-D `mesh` topology. The curves start on the bottom left corner of the topology and proceed along the lines in the red-orange-yellow-olive-green-blue order.

The `sweep` SFC mapping algorithm is the *straightforward* mapping that follows the nodes in the topology in a XYZ order. As *default reference mapping* we use `sweep` to allow the evaluation of the quality and performance improvements generated by all other mappings in *MapLib* (Section 7).

6.3 Communication- and Topology-Aware Mapping

`Bokhari` [17] is a mapping algorithm that originates in graph theory and was initially proposed for the assignment of parallel applications solving structural problems on the finite element machine. It takes an initial mapping and proceeds in two steps. In the first step, it searches for the pairwise task interchanges that maximize the cardinality (matching of edges of the application graph to edges in the machine topology) to generate a new mapping. In the second step, it evaluates if the new mapping has a higher cardinality than the previous one. If this is the case, it stores the best mapping, generates a new mapping through random task swaps, and returns to the first step. Otherwise, it returns the best mapping found previously.

The `topo-aware` [11] algorithm splits mapping into two phases. The first is a partitioning phase which groups heavily communicating processes in the same task. The second is the mapping phase in which tasks are mapped onto the processors such that more heavily communicating tasks are placed on nearby processors. The algorithm uses an estimation function that calculates the cost of placing an unallocated task on an available processor in each cycle.

We implemented three greedy algorithms: `greedy` [30], `FHgreedy` [21], and `greedyALLC` [26]. Both `greedy` and `FHgreedy` start mapping the most communicating process to a random node, while `greedyALLC` maps the most communicating process to the most connected node. Their main differences appear in the next steps they perform. `greedy` maps the heaviest communicating processes close to each other. `FHgreedy` maps neighbors according to the amount of communication that they have with each other. Lastly, `greedyALLC` first pairs the most communicating processes then maps them close to each other.

The `bipartition` mapping was proposed to improve inter-node mapping on `torus` and `mesh` topologies through a recursive bipartitioning algorithm [53]. The mapping is obtained by recursively dividing the application communication graph using a multilevel k-way partitioning algorithm [33] ($k = 2$), while the machine topology is simply recursively split in the middle of its largest dimension.

`PaCMap` [50] is a graph-based algorithm that simultaneously conducts job allocation and process mapping to reduce communication overhead. This work implements the process mapping step of `PaCMap`. It starts by partitioning the application communication matrix into process groups (PGs) of highly-communicating processes. After that, the algorithm selects a center PG (in this work, a single process) and maps it to a center node in the topology. Then, it expands the allocation by picking a node and mapping a process to it based on the network topology and the communication graph until all tasks are mapped. This results in highly-communicating PGs being mapped close to one another.

7 Performance Evaluation

In the proposed workflow, performance evaluation consists of three steps (green steps in Fig. 1). (i) The *pre-simulation performance analysis* step evaluates the mappings' quality using metrics derived without execution or simulation. (ii) The *post-simulation performance analysis* step evaluates the performance of the mappings using simulations. (iii) The *pre- and post-simulation performance comparison* shows the impact of mapping and allows the assertion of invariant properties (communication volume and exchanges) post-simulation.

Table 5 shows the parameters used in the design of the factorial experiments. The details and findings from these experiments are described in the following.

Table 5: Parameters used in the Design of Factorial Experiments

Parameter		**Values**	**Count**
Application		NAS CG, NAS BT-MZ CORAL2 AMG, CORAL LULESH	4
Mapping algorithm	Communication- & topology-oblivious	`Peano`, `Hilbert`, `Gray`, `sweep`, `scan`	5
	Communication- & topology-aware	`Bokhari`, `topo-aware`, `greedy`, `FHGreedy`, `greedyALLC`, `bipartition`, `PaCMap`	7
Mapping input		*commMatrix count*, *commMatrix size*	2
3-D topology		`mesh`, `torus`, `HAEC Box`	3
Communication model		NCD_r	1
Total factorial experiments			**288**

7.1 Pre-simulation Mapping Performance Analysis

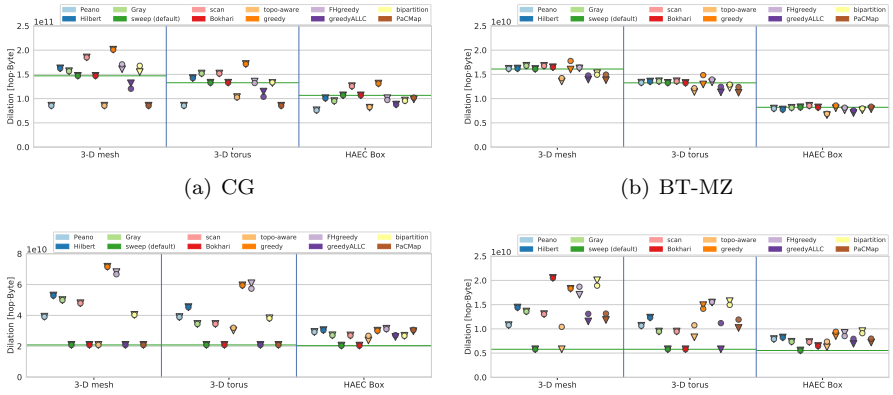
The *pre-simulation analysis* step evaluates the mappings' quality using metrics derived without execution or simulation. This step requires the following information: the application communication behavior, denoted by *commMatrix count* and *commMatrix size*, the process mappings, the specification of the target network topology, and the routing algorithm.

Given this information, one can use metrics such as dilation, average and total number of hops traveled by the application messages, and volume communication transmitted through the network links [54].

In this work, we use *dilation* as a pre-simulation metric to evaluate the mappings' quality. Dilation is widely used as a performance and quality metric in process mapping [11, 18, 31, 54], where it is also referred to as 'hop-Bytes'. Dilation is easy to determine and tends to correlate well with the applications' performance [15]. Dilation, denoted as D, can be calculated using (1), where P denotes the set of application processes, δ denotes the mapping function, d represents the distance between nodes (number of hops), and w denotes the weight function (e.g., the communication volume in Byte). *A lower dilation value leads to improved performance and implicitly to a communication energy consumption reduction.*

$$\mathrm{D} = \sum_{i \in P} \sum_{j \in P} d(\delta(i), \delta(j)) \cdot w(i, j). \tag{1}$$

Fig. 4 shows the dilation associated with each mapping for every application–topology pair. One can note that dilation significantly varies across topologies and mappings. Specifically, most mappings improved performance over `sweep` (*default*) for CG and BT-MZ.



(a) CG (b) BT-MZ

(c) LULESH (d) AMG

Figure 4: Dilation for all applications, mappings, and topologies. Topologies are shown on the X axis. The dilation values are shown on the Y axis. Each mapping is represented with a distinct color. The green horizontal line denotes the dilation of the `sweep` mapping, which we consider as the default mapping on a given topology. Circles ∘ denote mappings based on *commMatrix count*, while triangles ▽ indicate mappings based on *commMatrix size*.

CG achieves the lowest dilation with `Peano`, `topo-aware`, and `PaCMap` for 3-D `mesh`, with `Peano` and `PaCMap` for 3-D `torus`, and with `Peano` for `HAEC Box`. BT-MZ achieves the lowest dilation with `topo-aware` for 3-D `mesh` and `HAEC Box`. For `torus`, the `PaCMap` mapping with *commMatrix size* achieved the lowest dilation. These observations indicate that CG and BT-MZ exhibit the potential for performance improvement with these mappings. LULESH and AMG achieved the lowest dilation with `sweep` (default) which indicates that they will not benefit from any of the other mappings on these topologies.

These results also illustrate the sensitivity of the performance of mapping algorithms to the application and topology. For instance, `Bokhari` shows the highest dilation for AMG on the 3-D `mesh` but one of lowest ones on the 3-D `torus`.

Even though the dilation of the mappings performed using *commMatrix count* and *commMatrix size* is comparable, *commMatrix size* helped mappings achieve a slightly lower dilation in most cases. *This indicates that commMatrix size is a mapping input that has a greater impact on performance.*

Directly comparing a mapping algorithm across the topologies, the performance on the `HAEC Box` topology always yields the lowest dilation due to its increased link connectivity which offers shorter paths for messages to travel on.

7.2 Application Performance Prediction Using Simulation

We use HAEC-SIM [16] [4] to simulate the applications' performance. HAEC-SIM uses an application's trace as an application model during the simulation process. With the respective models for the target topology, the simulator generates new traces representing the applications' predicted behavior on the target system. The simulator works deterministically, i.e., simulations with the same input will repeatedly generate the same output. HAEC-SIM's parameters are the application trace, the mapping, and the configuration file specifying the target system [37].

While the duration of the computation operations are fixed in the input trace, we use a contention-oblivious implementation of the NCD_r network model [44] to model the point-to-point communications and their predicted duration on the target topology with the given link characteristics. Pfennig et al. verified this implementation in [43]. While HAEC-SIM uses sophisticated modeling of point-to-point communications (based on network coding), the model for collective communications adds a fixed minimum delay and serves as a temporal synchronization of all involved processes during simulation.

To predict the performance impact of the different mapping algorithms, we simulate the execution for each combination of application, mapping, and topology (see Table 5). We analyze the resulting HAEC-SIM application traces post-simulation. The configuration files used for the simulations can by found in Appendix A.1

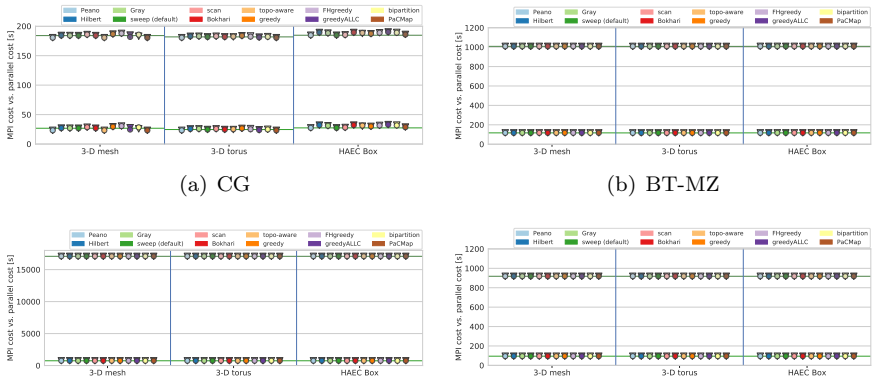
7.3 Post-simulation Performance Evaluation

We use the simulation results (traces) to calculate the communication model time, the cost of MPI point-to-point communications, the overall parallel cost of the application [27], as well as post-simulation communication metrics: volume, distance, and dilation.

The communication model time is the *sum* of the duration of all point-to-point message transfers on the transport layer according to the NCD_r communication model. The MPI point-to-point cost denotes the *aggregated* time that all application processes spent in MPI point-to-point communication functions, such as `MPI_Send`, `MPI_recv`, and `MPI_Wait`.

Fig. 5 shows the impact of process mapping on application performance. The plots present the values of the application's *parallel cost* (calculated as parallel wall time multiplied by the number of nodes) and the part attributed to the *aggregated* MPI point-to-point costs.

These results show that, except for CG, neither the topology nor the mapping impact the values of the above performance metrics. For CG, the 3-D `torus` topology offers the highest performance, in particular with `Peano` and `PaCMap`.



(a) CG (b) BT-MZ

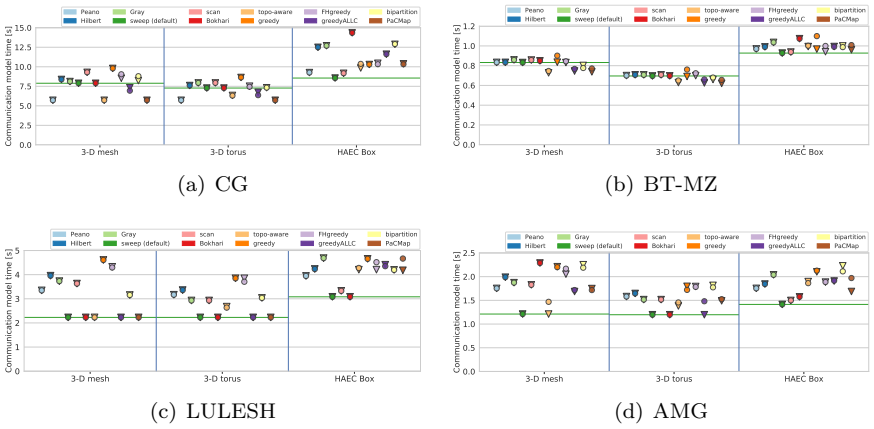
(c) LULESH (d) AMG

Figure 5: Simulated parallel execution cost and parallel communication cost for all applications, mappings, and topologies. Topologies are shown on the X axis. The lower range on the Y axis shows the MPI point-to-point cost while the upper range on the Y axis shows the parallel cost. Each mapping is represented with a distinct color. Circles $\circ$ denote mappings based on *commMatrix count*, while triangles $\triangledown$ indicate mappings based on *commMatrix size*. The green horizontal lines delineate the performance achieved by `sweep` for both cost metrics.

Additionally, for the 3-D `mesh` topology, the `topo-aware` mapping yields high performance. On the `HAEC Box`, `sweep` has the lowest MPI and parallel costs.

However, a different conclusion becomes visible, when we look at the communication model time in Fig. 6. This figure shows the costs of the communication from the network perspective and reveals significant differences in the communication time. One can notice that the `HAEC Box` shows the highest minimum times among the topologies in Fig. 6. This is an effect of its wireless links that possess higher latencies (Table 4). In this situation, `sweep` and `scan` exhibit the highest performance as they map processes contiguously in the same XY plane (Fig. 3). *This observation highlights the need for process mapping algorithms tailored towards heterogeneous network topologies.*

While the mappings may not have influenced the application run-time in these simulations, *the communication model time changed significantly.* Therefore, at the network level, poor mappings lead to increased power consumption and a higher probability of contention, which are known to lead to significant performance loss.



(a) CG (b) BT-MZ

(c) LULESH (d) AMG

Figure 6: Communication model time for all applications, mappings, and topologies. Topologies are shown on the X axis. The Y axis shows the sum of the transmission time over the transport layer according to the NCD_r model. The distinct colors represent the mappings. Circles $\circ$ denote mappings based on *commMatrix count*, while triangles $\triangledown$ indicate mappings based on *commMatrix size*. The green horizontal lines delineate the performance achieved by `sweep`.

7.4 Pre- and Post-simulation Performance Comparison

Based on the knowledge about the application prior to simulation, we infer certain assertions about the predicted performance of the applications and use them to verify the simulation results. According to the definition of dilation, its pre-simulation values cannot differ from its post-simulation values. As part of the automated simulation process, we calculate this metric before and after each simulation experiment and assert that it remains unchanged.

All mapping algorithms use *commMatrix count* and *commMatrix size* as input (see Section 4.2). This duality results in two mappings for every application–mapping–system configuration. Due to the nature of the communication-oblivious process mapping algorithms, we effectively generate the same mapping twice, i.e., the output based on *commMatrix count* is equal to the one for *commMatrix size*. As the simulation is deterministic, the simulated performance is expected to be equal for these mapping tuples. Fig. 5 illustrates and confirms this, which increase the confidence in the results in both Fig. 5 and Fig. 6.

In the *pre-simulation* performance evaluation (Section 7.1), we measure the dilation for the different applications–mappings–topologies configurations. Given the significant differences in the dilation, this metric hints at great performance optimization opportunities. However, except for the CG application, one can only see minor differences in the simulated parallel costs (Fig. 5). This behavior stems from several intricacies, which are discussed in the following.

As shown in Table 1, CG is the only application that frequently uses blocking MPI point-to-point operations. In combination with the high-speed future-

generation network links (see Table 4), the other applications successfully mask their communication costs from being measurable from the application perspective by using non-blocking MPI point-to-point operations. Moreover, the communication volume in CG is an order of magnitude higher than that of the other applications highlighting the effect of the communication masking even further.

Furthermore, since NCD_r is a contention-oblivious communication model, concurrent communications do not content for resources and, therefore, no performance degradation can be observed without explicitly modeling congestion (part of future work).

The comparison of *pre-simulation* dilation against the corresponding MPI point-to-point communication time, leads to new insights. The communication times differences in Fig. 6 are in agreement with the performance prediction derived from the dilation for all applications mapped to the 3-D `mesh` and 3-D `torus` topology. For the `HAEC Box` topology, no such correlation can be made. *This shows that the dilation metric needs a parameter to distinguish hops with different characteristics that are prevalent in heterogeneous networks.*

These observations confirm that *dilation is suitable as a prediction for the communication time on homogeneous topologies. They also reveal the need for a novel performance metric for process mapping on heterogeneous topologies.*

8 Conclusions and Future Work

In this work we showed that process mapping is an important application optimization step that should be performed at every execution. In a real setting, our work can be practiced first by binding the processing elements to their dedicated hardware processing units and subsequently by reordering the MPI ranks of the application according to the relevant mapping algorithm.

We proposed and exercised a generic workflow to support process mapping as an explicit application optimization step. Using trace-based simulation, we predicted the communication performance for four applications, mapped with twelve mapping algorithms (implemented as a Python library), executing on three topologies. Using pre- and post-simulation metrics, we showed that communication-aware mappings frequently outperform communication-oblivious ones.

We observed that dilation (measured as hop·Byte) does not correlate well with the simulated application execution time. However, when looking exclusively at the aggregated communication times, performance varies significantly among the mappings. This observation indicates that even if the application execution time does not change, a careful mapping reduces the network load. Therefore, *mapping matters.*

In future work, our *MapLib* library can be augmented by other mapping algorithms, such as those from Bhatele et al. [15] and Wu et al. [52]. We see that HAEC-SIM could also be integrated with TopGen [51] to simulate additional interconnection networks and a contention-aware communication model. Inter-

application interference and the impact of process mapping on other metrics (such as congestion) are also part of future work. A scalability study considering topologies with higher dimensions and a greater number of processes is also an important aspect to be considered. Finally, the impact of the combination of application-aware routing algorithms with communication-aware mapping algorithms is an interesting research question.

Acknowledgments

This work is in part supported by the Swiss National Science Foundation in the context of the "Multi-level Scheduling in Large Scale High Performance Computers" (MLS) grant, number 169123 and by the German Research Foundation (DFG) within the CRC912 - HAEC. The authors acknowledge Daniel Besmer and Viacheslav Sharunov for their earlier contribution to this work.

References

[1] Application 352.nab from the SPEC OpenMP 2012 Benchmarks. https://www.nas.nasa.gov/publications/npb_problem_sizes.html. Accessed: May 10, 2020.

[2] Application AMG from CORAL2 Benchmarks. https://asc.llnl.gov/coral-2-benchmarks/downloads/AMG_Summary_v1_7.pdf. Accessed: May 09, 2020.

[3] Application LULESH from CORAL benchmarks. https://computing.llnl.gov/projects/co-design/lulesh. Accessed: May 09, 2020.

[4] HAEC simulator Website. https://tu-dresden.de/zih/haec_sim. Accessed: May 19, 2020.

[5] Intel Broadwell E5-2640 v4. https://ark.intel.com/content/\www/us/en/ark/products/92984/intel-xeon-processor-e5-2640-v4-25m-cache-2-40-ghz.html. Accessed: April 18, 2020.

[6] MapLib GitHub Repository. https://github.com/unibas-dmi-hpc/MapLib. Accessed: May 20, 2020.

[7] Process-logical communication matrix-based statistics scripts github repository. https://github.com/llpilla/communication-statistics. Accessed: May 20, 2020.

[8] The CORAL Benchmarks Suite. https://asc.llnl.gov/CORAL-benchmarks/. Accessed: May 09, 2020.

[9] The CORAL2 Benchmarks Suite. https://asc.llnl.gov/coral-2-benchmarks/. Accessed: May 09, 2020.

[10] The NAS Parallel Benchmarks Suite. https://www.nas.nasa.gov/publications/npb.html. Accessed: May 09, 2020.

[11] Agarwal, T., Sharma, A., Laxmikant, A., and Kale, L. V. Topology-aware task mapping for reducing communication contention on large parallel machines. In *Proceedings 20th IEEE International Parallel Distributed Processing Symposium* (April 2006).

[12] Ait Salaht, F., Desprez, F., and Lebre, A. An overview of service placement problem in fog and edge computing. *ACM Comput. Surv. 0*, ja (2020).

[13] Bader, M. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer Heidelberg New York Dordrecht London, 2013.

[14] Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., et al. The nas parallel benchmarks. *The International Journal of Supercomputing Applications 5*, 3 (1991), 63–73.

[15] Bhatelé, A., Gupta, G. R., Kalé, L. V., and Chung, I. Automated mapping of regular communication graphs on mesh interconnects. In *2010 International Conference on High Performance Computing* (2010), pp. 1–10.

[16] Bielert, M., Ciorba, F. M., Feldhoff, K., Ilsche, T., and Nagel, W. E. HAEC-SIM: A simulation framework for highly adaptive energy-efficient computing platforms. In *Eighth EIA International Conference on Simulation Tools and Techniques* (Aug. 2015), G. Theodoropoulos, G. S. H. Tan, and C. Szabo, Eds., ACM.

[17] Bokhari, S. H. On the mapping problem. *IEEE Transactions on Computers*, 3 (1981), 207–214.

[18] Bordage, C., and Jeannot, E. Process affinity, metrics and impact on performance: An empirical study. In *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing* (2018), CCGrid '18, IEEE Press, pp. 523–532.

[19] Chen, H., Chen, W., Huang, J., Robert, B., and Kuhn, H. MPIPP: An Automatic Profile-Guided Parallel Process Placement Toolset for SMP Clusters and Multiclusters. In *Proceedings of the 20th Annual International Conference on Supercomputing* (New York, NY, USA, 2006), ICS '06, Association for Computing Machinery, pp. 353–360.

[20] Cruz, E. H. M., Diener, M., Pilla, L. L., and Navaux, P. O. A. Eagermap: A task mapping algorithm to improve communication and load

balancing in clusters of multicore systems. *ACM Trans. Parallel Comput. 5*, 4 (Mar. 2019).

[21] Deveci, M., Kaya, K., Uçar, B., and ÇatalyÜrek, Ü. V. Fast and high quality topology-aware task mapping. In *2015 IEEE International Parallel and Distributed Processing Symposium* (2015), IEEE, pp. 197–206.

[22] Diener, M., Cruz, E. H., Alves, M. A., Alhakeem, M. S., Navaux, P. O., and Heiss, H.-U. Locality and balance for communication-aware thread mapping in multicore systems. In *European Conference on Parallel Processing* (2015), Springer, pp. 196–208.

[23] Diener, M., Cruz, E. H., Pilla, L. L., Dupros, F., and Navaux, P. O. Characterizing communication and page usage of parallel applications for thread and data mapping. *Performance Evaluation 88* (2015), 18–36.

[24] European Technology Platform for High Performance Computing (ETP4HPC). SRA4: Strategic Research Agenda for High-Performance Computing in Europe.

[25] Fettweis, G., Nagel, W., and Lehner, W. Pathways to servers of the future. In *2012 Design, Automation Test in Europe Conference Exhibition (DATE)* (March 2012), pp. 1161–1166.

[26] Glantz, R., Meyerhenke, H., and Noe, A. Algorithms for mapping parallel processes onto grid and torus architectures. In *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing* (March 2015), pp. 236–243.

[27] Grama, A., Gupta, A., Karypis, G., and Kumar, V. *Introduction to Parallel Computing*, second ed. Pearson, 2015.

[28] Hilbert, D. V. Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen* (1891), 459–460.

[29] Hoefler, T., Jeannot, E., and Mercier, G. An Overview of Process Mapping Techniques and Algorithms in High-Performance Computing. In *High Performance Computing on Complex Environments*, E. Jeannot and J. Zilinskas, Eds. Wiley, June 2014, pp. 75–94.

[30] Hoefler, T., and Snir, M. Generic topology mapping strategies for large-scale parallel architectures. In *Proceedings of the International Conference on Supercomputing* (New York, NY, USA, 2011), ICS '11, ACM, pp. 75–84.

[31] Jain, N., Bhatele, A., Robson, M. P., Gamblin, T., and Kale, L. V. Predicting application performance using supervised learning on communication features. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2013), SC '13, Association for Computing Machinery.

[32] Karlin, I., Bhatele, A., Keasler, J., Chamberlain, B. L., Cohen, J., DeVito, Z., Haque, R., Laney, D., Luke, E., Wang, F., Richards, D., Schulz, M., and Still, C. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)* (Boston, USA, may 2013).

[33] Karypis, G., and Kumar, V. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing 48*, 1 (1998), 96–129.

[34] Kenny, J. P., Sargsyan, K., Knight, S., Michelogiannakis, G., and Wilke, J. J. The pitfalls of provisioning exascale networks: A trace replay analysis for understanding communication performance. In *High Performance Computing* (Cham, 2018), R. Yokota, M. Weiland, D. Keyes, and C. Trinitis, Eds., Springer International Publishing, pp. 269–288.

[35] Kienhuis, B., Deprettere, E. F., van der Wolf, P., and Vissers, K. *A Methodology to Design Programmable Embedded Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 18–37.

[36] Knüpfer, A., Rössel, C., Mey, D. a., Biersdorff, S., Diethelm, K., Eschweiler, D., Geimer, M., Gerndt, M., Lorenz, D., Malony, A., Nagel, W. E., Oleynik, Y., Philippen, P., Saviankou, P., Schmidl, D., Shende, S., Tschüter, R., Wagner, M., Wesarg, B., and Wolf, F. Score-P: A joint performance measurement run-time infrastructure for Periscope, Scalasca, TAU, and Vampir. In *Tools for High Performance Computing 2011* (Berlin, Heidelberg, 2012), H. Brunst, M. S. Müller, W. E. Nagel, and M. M. Resch, Eds., Springer Berlin Heidelberg, pp. 79–91.

[37] Korndörfer, J. H. M., Bielert, M., Pilla, L. L., and Ciorba, F. M. Mapping matters: Application process mapping on 3-d processor topologies. https://arxiv.org/abs/2005.10413, 2020.

[38] Laguna, I., Marshall, R., Mohror, K., Ruefenacht, M., Skjellum, A., and Sultana, N. A large-scale study of mpi usage in open-source hpc applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (New York, NY, USA, 2019), SC '19, Association for Computing Machinery.

[39] Mahmud, R., Srirama, S. N., Ramamohanarao, K., and Buyya, R. Profit-aware application placement for integrated fog–cloud computing environments. *Journal of Parallel and Distributed Computing 135* (2020), 177 – 190.

[40] Mercier, G., and Clet-Ortega, J. Towards an efficient process placement policy for mpi applications in multicore environments. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*

(Berlin, Heidelberg, 2009), M. Ropo, J. Westerholm, and J. Dongarra, Eds., Springer Berlin Heidelberg, pp. 104–115.

[41] Mokbel, M. F., Aref, W. G., and Kamel, I. Analysis of multi-dimensional space-filling curves. *GeoInformatica 7*, 3 (2003), 179–209.

[42] Peano, G. Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen* (1890), 157–160.

[43] Pfennig, S., Feldhoff, K., Ciorba, F. M., Bielert, M., Franz, E., Ilsche, T., Reiher, T., and Nagel, W. E. Simulation models verification for resilient communication on a highly adaptive energy-efficient computer. In *Proceedings of the 24th High Performance Computing Symposium (HPC 2016), part of the 2016 Spring Simulation Multi-Conference, SpringSim'16, CA, USA, April 3-6, 2016* (San Diego, CA, USA, Apr. 2016), HPC '16, Society for Computer Simulation International, pp. 6:1–6:8. April 03 - 06, 2016.

[44] Pfennig, S., and Franz, E. Adjustable redundancy for secure network coding in a unicast scenario. In *2014 International Symposium on Network Coding (NetCod)* (2014), pp. 1–6.

[45] Rico-Gallego, J. A., Díaz-Martín, J. C., Manumachu, R. R., and Lastovetsky, A. L. A survey of communication performance models for high-performance computing. *ACM Comput. Surv. 51*, 6 (Jan. 2019).

[46] Rodrigues, E. R., Madruga, F. L., Navaux, P. O. A., and Panetta, J. Multi-core aware process mapping and its impact on communication overhead of parallel applications. In *2009 IEEE Symposium on Computers and Communications* (2009), pp. 811–817.

[47] Schöne, R., Ilsche, T., Bielert, M., Gocht, A., and Hackenberg, D. Energy efficiency features of the intel skylake-sp processor and their impact on performance. *arXiv preprint arXiv:1905.12468* (2019).

[48] Sensi, D. D., Girolamo, S. D., and Hoefler, T. Mitigating network noise on dragonfly networks through application-aware routing. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2019, Denver, Colorado, USA, November 17-19, 2019* (2019), M. Taufer, P. Balaji, and A. J. Peña, Eds., ACM, pp. 16:1–16:32.

[49] Tsuji, M., Boku, T., and Sato, M. Scalable communication performance prediction using auto-generated pseudo mpi event trace. In *Proceedings of the International Conference on High Performance Computing in Asia-Pacific Region* (New York, NY, USA, 2019), HPC Asia 2019, Association for Computing Machinery, pp. 53–62.

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

[51] Villar, J. A., Mathey, G. M., Escudero-Sahuquillo, J., Garcia, P. J., Alfaro, F. J., Sanchez, J. L., and Quiles, F. J. TopGen: A library to provide simulation tools with the modeling of interconnection network topologies. In *2018 International Conference on High Performance Computing Simulation (HPCS)* (2018), pp. 452–459.

[52] Wu, J., Xiong, X., Berrocal, E., Wang, J., and Lan, Z. Topology mapping of irregular parallel applications on torus-connected supercomputers. *The Journal of Supercomputing 73*, 4 (2017), 1691–1714.

[53] Wu, J., Xiong, X., and Lan, Z. Hierarchical task mapping for parallel applications on supercomputers. *The Journal of Supercomputing 71*, 5 (May 2015), 1776–1802.

[54] Yu, H., Chung, I.-H., and Moreira, J. Topology mapping for blue gene/l supercomputer. In *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing* (New York, NY, USA, 2006), SC '06, Association for Computing Machinery, pp. 116–es.

A Appendix

A.1 HAEC-SIM configuration files

Listing 1: HAEC-SIM configuration file for the 3-D `mesh` topology

```
{
  "platform": {
    "link_types": [
      {
        "name" : "optical",
        "bandwidth" : 31250000000,
        "latency" : 10,
        "bit_error_rate": 1e-12,
        "packet_attack_rate": 0.01
      },
      {
        "name" : "wireless",
        "bandwidth" : 12500000000,
        "latency" : 100,
        "bit_error_rate": 1.0e-8,
        "packet_attack_rate": 0.01
      }
    ],
    "topology": {
      "shape": [ 4, 4, 4 ],
      "link_types": [ "optical", "optical", "optical" ],
      "connectivity": [ "mesh", "mesh", "mesh" ]
    }
  },
  "modules": {
    "static_network_model": {
      "routing": "shortest_path",
      "parameter_search_folders": "haec-sim/share/modules/static_network_model/parameters",
      "communication_model": "DNCr",
      "size_packet": 1500,
      "size_finitefield": 8,
      "size_window": 5,
      "delay_processing": 63,
      "delay_mpi": 5,
      "size_mpi_header": 16,
      "size_windowid": 4,
      "size_packetid": 2,
      "size_signature": 256,
```

```
      "size_generationid": 4
    }
  }
}
```

Listing 2: HAEC-SIM configuration file for the 3-D `torus` topology

```
{
  "platform": {
    "link_types": [
      {
        "name" : "optical",
        "bandwidth" : 31250000000,
        "latency" : 10,
        "bit_error_rate": 1e-12,
        "packet_attack_rate": 0.01
      },
      {
        "name" : "wireless",
        "bandwidth" : 12500000000,
        "latency" : 100,
        "bit_error_rate": 1.0e-8,
        "packet_attack_rate": 0.01
      }
    ],
    "topology": {
      "shape": [ 4, 4, 4 ],
      "link_types": [ "optical", "optical", "optical" ],
      "connectivity": [ "torus", "torus", "torus" ]
    }
  },
  "modules": {
    "static_network_model": {
      "routing": "shortest_path",
      "parameter_search_folders": "haec-sim/share/modules/static_network_model/parameters",
      "communication_model": "DNCr",
      "size_packet": 1500,
      "size_finitefield": 8,
      "size_window": 5,
      "delay_processing": 63,
      "delay_mpi": 5,
      "size_mpi_header": 16,
      "size_windowid": 4,
      "size_packetid": 2,
```

```
      "size_signature": 256,
      "size_generationid": 4
    }
  }
}
```

Listing 3: HAEC-SIM configuration file for the 3-D `haec` topology

```
{
  "platform": {
    "link_types": [
      {
        "name" : "optical",
        "bandwidth" : 31250000000,
        "latency" : 10,
        "bit_error_rate": 1e-12,
        "packet_attack_rate": 0.01
      },
      {
        "name" : "wireless",
        "bandwidth" : 12500000000,
        "latency" : 100,
        "bit_error_rate": 1.0e-8,
        "packet_attack_rate": 0.01
      }
    ],
    "topology": {
      "shape": [ 4, 4, 4 ],
      "link_types": [ "optical", "optical", "wireless" ],
      "connectivity": [ "torus", "torus", "crossbar" ]
    }
  },
  "modules": {
    "static_network_model": {
      "routing": "haec_box",
      "parameter_search_folders": "haec-sim/share/modules/static_network_model/parameters",
      "communication_model": "DNCr",
      "size_packet": 1500,
      "size_finitefield": 8,
      "size_window": 5,
      "delay_processing": 63,
      "delay_mpi": 5,
      "size_mpi_header": 16,
      "size_windowid": 4,
```

```
        "size_packetid": 2,
        "size_signature": 256,
        "size_generationid": 4
    }
  }
}
```