MAGC: A Mapping Approach for GPU Clusters

Seyed H. Mirsadeghi, Iman Faraji, and Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University, Kingston, ON, Canada K7L 3N6
Email: {s.mirsadeghi, i.faraji, ahmad.afsahi}@queensu.ca

***Abstract*—GPU accelerators have been increasingly used in modern heterogeneous HPC clusters by offering high performance and energy efficiency. Such heterogeneous GPU clusters consisting of multiple CPU cores and GPU devices have become the platform of choice for many HPC applications. The communication channels among these processing elements expose different latency and bandwidth characteristics. Thus, efficient utilization of communication channels becomes an important factor for achieving higher inter-process communication performance.**

In this paper, we exploit topology awareness for a better utilization of communication channels in GPU clusters. We first discuss the challenges associated with topology-aware mapping in GPU clusters, and then propose MAGC, a Mapping Approach for GPU Clusters. MAGC seeks to improve the total communication performance by a joint consideration of both CPU-to-CPU and GPU-to-GPU communications of the application, and CPU and GPU physical topologies of the underlying GPU cluster. It provides a unified framework for topology-aware process-to-core mapping and GPU-to-process assignment across a GPU cluster. We study the potential benefits of MAGC with two different mapping algorithms: a) the Scotch graph mapping library, and b) a heuristic designed to explicitly consider maximum congestion.

We evaluate our design through extensive experiments at micro-benchmark and application levels on two GPU clusters with different GPU types and topologies. We have developed a micro-benchmark suite to model various communication patterns among CPU cores and among GPU devices. For application results, we use the molecular dynamics simulator, HOOMD-blue. Micro-benchmark results show that we can achieve up to 91.4% improvement in communication time. At the application level, we can achieve up to 8% performance improvement.

***Index Terms*—GPU cluster; Topology awareness; GPU selection; Mapping.**

I. Introduction

Over the past decade, the high-performance computing (HPC) landscape has changed significantly, particularly due to the emergence of accelerators. In particular, GPU accelerators have established themselves in modern heterogeneous HPC clusters by offering high performance and energy efficiency. To further increase the computational power and tackle larger problems, such systems provide various processing elements (PEs) including multiple CPUs each having multiple cores, as well as multiple GPU devices. The communication channels among these PEs are commonly organized in a hierarchical fashion. Such a design inherently induces some levels of heterogeneity in terms of the performance of communication channels among different PEs. The inter-process communication among PEs will pass through different communication channels with different latency and bandwidth characteristics. Thus, efficient utilization of communication channels becomes an important factor for achieving higher inter-process communication performance in such systems.

One way to address the issue mentioned above is through topology awareness. A topology-aware assignment of processing elements to application processes can lead to a more efficient utilization of communication channels. The benefits have already been shown for CPU-to-CPU communications in previous studies [1]–[4]. In addition, in our previous work [5], we showed that similar issues exist for GPU-to-GPU communications within multi-GPU nodes as well, and for that we proposed a topology-aware GPU selection scheme to enhance the communication performance.

In this work, we study the problem for GPU clusters consisting of multiple multi-GPU nodes where we will have a joint problem of process-to-core(node) mapping and GPU-to-process assignment. More specifically, we seek to answer the following question: Given an application that uses both CPU and GPU processing elements, how should such PEs be assigned to application processes so that the ultimate communication performance is optimized? In essence, the question is which core/node should host each process, and which GPU should be assigned to it. This is a challenging problem because optimizing for one communication pattern (e.g., CPU-to-CPU) could conflict with optimization for another (i.e., GPU-to-GPU). Moreover, the CPUs and GPUs could have different communication patterns, as well as different intra-node physical topologies.

In this paper, we first elaborate on how process-to-CPU mapping and GPU-to-process assignment become related to each other (and impact each other) in GPU clusters, and then discuss how a unified approach can be used to address both problems. More specifically, we propose MAGC, a Mapping Approach for GPU Clusters. MAGC attempts to improve the total communication performance by considering both CPU-to-CPU and GPU-to-GPU communications of an application[1], as well as the CPU and GPU physical topologies of the target cluster. MAGC exploits a three-phase approach that first assigns processes to the nodes across the network, and then performs CPU core bindings and GPU assignments within each node. To the best of our knowledge, this is the first work that studies topology-aware mapping for GPU clusters, and proposes a unified methodology for process-to-CPU mapping and GPU-to-process assignment. We also study MAGC's ben-

[1]We interchangeably use the terms CPU-to-CPU and CPU communications to refer to the communications among the CPU cores of a cluster. Same thing applies to GPU-to-GPU and GPU communications for GPU devices.

efits with two different algorithms used for finding the desired core bindings and GPU assignments within each node. The first algorithm uses the Scotch library [6], whereas for the second one we have designed a congestion-based heuristic. The heuristic uses the maximum congestion imposed on the intra-node communication channels as the metric to find the desired mappings.

We evaluate our design through extensive experiments at micro-benchmark and application levels on two clusters with different GPU types and topologies. We have developed a micro-benchmark suite to model various communication patterns among CPU cores and among GPU devices. The results show that MAGC can successfully improve performance for a wide variety of micro-benchmarks by up to 91.4%. For application results, we use the molecular dynamics simulator, HOOMD-blue [7]. The results show that we can achieve up to 8% performance improvement for HOOMD-blue.

The rest of the paper is organized as follows. In Section II, we provide some background information. Section III reviews the related work. We discuss the problem details in Section IV. Section V is devoted to our proposed framework and mapping heuristic. Experimental evaluations are covered in Section VI, followed by conclusion and future work in Section VII.

II. Background

A. GPU-Aware MPI

The Message Passing Interface (MPI) [8] is the de-facto standard for parallel programming. It provides support for various types of communications such as point-to-point, collective, and one-sided. Support for GPU communications has been added to well-known MPI implementations such as MVAPICH2 [9] and Open MPI [10]. It may follow a general approach which involves staging GPU data into the host buffer and leveraging the traditional CPU-based MPI routines. It may also involve further tunings by pipelining the transfers and using specifically designed algorithms. In addition, some hardware-related features such as CUDA IPC and GPUDirect RDMA can provide direct GPU-to-GPU communications.

B. Topology-Aware Mapping

In general, topology-aware mapping is an instance of the graph mapping problem where a guest graph G is mapped onto a host graph H. In the parallel computing context, G represents the communication pattern of the application, whereas H models the physical topology of the target system. The problem is well known to be NP-hard and hence, practical solutions involve various heuristics that attempt to find sub-optimal solutions. One commonly used approach for solving the problem is through graph partitioning. There exist various libraries/tools such as Scotch [6] and METIS [11] that provide APIs and algorithms for partitioning graphs into a balanced subset of vertices with minimum edge-cut between them. Using such libraries, one can recursively partition G and H into a subset of vertices, and map each subset in G onto a peer subset in H. The Scotch library in particular provides a direct graph mapping interface for this purpose.

C. Topology Information

The communication pattern of an application can be extracted by profiling communications in an initial run. More specifically, instrumentation at the MPI library level can be used to log all message transfers among the processes. The results are stored in terms of a matrix that captures the total volume of communication among all MPI ranks. With an appropriate set of tools, it will also be possible to extract such information through the new `MPI_T` interface [12] introduced in MPI 3.0. On the other hand, information about the physical topology of the target system can be extracted by various tools. At the network layer, the topology can be queried by APIs provided by the underlying interconnect. In InfiniBand, `ibnetdiscover` can be used to extract the topology. Researchers have also designed higher-level topology detection services [13]. At the intra-node layer, the hardware architecture can be extracted by libraries such as HWLOC [14] and/or NVML (NVIDIA Management Library) [15]. HWLOC provides portable APIs to query various attributes of a node including cores, sockets, caches, etc. NVML provides a set of APIs for extracting various characteristics of NVIDIA GPU devices, including the physical topology of GPUs within a multi-GPU node.

III. Related Work

Topology-aware mapping has been extensively studied in the context of communication in traditional HPC systems. Several experiments on large-scale systems such as IBM BG/P and Cray XT have verified the adverse effects of contention and hop-count on message latencies [1]. Another study [2] shows that different mappings of an application on large-scale IBM BG/P systems can significantly affect performance. Accordingly, Bhatelé and Kalé [3] propose several heuristics for mapping communication graphs onto mesh topologies.

Hoefler and Snir [16] propose generic mapping heuristics for arbitrary communication patterns and interconnection networks. The heuristics generally attempt to map communication graphs onto network topology graphs so that highly communicating processes fall closer to each other within the network. In [17], the authors propose PTRAM, which is a parallel topology- and routing-aware mapping framework. It consists of a greedy mapping heuristic and a mapping refinement algorithm. The former attempts to minimize a hybrid metric that can evaluate a mapping from different aspects, whereas the latter attempts to reduce maximum congestion directly.

In [18], the authors propose a topology-aware mapping mechanism for MPI topology functions. The mapping is done by using the Scotch library. Jeannot and Mércier [4] propose a mapping approach for intra-node process-to-core binding. The hardware architecture is modeled by a tree in compliance with the hierarchical architecture of NUMA nodes. An algorithm called TreeMatch is used to do the actual mapping.

In [5], the authors show the importance of topology awareness for GPU communications in a single multi-GPU node. It is shown that the performance of different intra-node GPU communication channels can be considerably different from

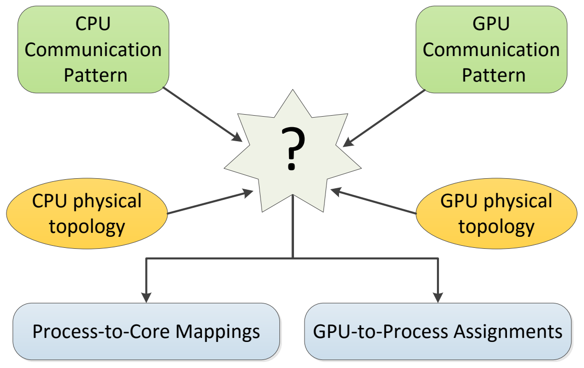

CPU Communication Pattern
GPU Communication Pattern
?
CPU physical topology
GPU physical topology
Process-to-Core Mappings
GPU-to-Process Assignments

Figure 1: Schematic diagram of the joint process-to-core mapping and GPU-to-process assignment problem.

each other. Accordingly, the authors propose a topology-aware GPU selection scheme to efficiently assign GPUs to MPI processes. To the best of our knowledge, [5] is the first work that considers topology awareness for intra-node GPU communications. In [19], the authors propose a congestion-aware performance model for the PCIe fabric used to connect GPUs in a multi-GPU node. The model can help to design better algorithms for intra-node GPU communications.

IV. Problem Statement

In heterogeneous GPU clusters, topology-aware mapping is a joint problem of process-to-core(node) mapping and GPU-to-process assignment. Fig. 1 illustrates a schematic diagram of this problem where we have the CPU and GPU communication patterns, as well as the CPU and GPU topologies as input, and would like to find an efficient process-to-core mapping and GPU-to-process assignment.

One way to address the joint problem is to independently figure out process-to-core mappings and GPU-to-process assignments in separate steps. For instance, one could first bind processes to CPU cores based on the CPU communication pattern and CPU physical topology of the system, and then assign GPUs to processes with respect to the GPU communication pattern and the physical topology of the GPUs. At each step, a mapping algorithm/tool (e.g., Scotch) can be used to find an optimized assignment. As discussed in Section III, the first step of such an approach (i.e., process-to-core mapping) has extensively been studied before. Moreover, in [5] we studied topology-aware GPU assignment at a single node level. However, in the following, we will discuss the shortcomings of such an approach for addressing the problem shown in Fig. 1, and will motivate the need for a unified topology-aware mapping framework.

Deriving process-to-core mapping and GPU assignment independently could easily lead to a poor result because a particular process-to-core mapping could limit choices for efficient assignment of GPUs. In fact, the two steps could lead to conflicting process placement strategies. This is specially true when we consider GPU assignment across multiple nodes. For instance, consider two processes with zero (or low) CPU communications between them. Topology-aware mapping strategies would typically map such two processes on nodes that are far from each other. However, the same two processes could have a high amount of GPU communications with each other which necessitates to assign them to two GPUs that are physically close to each other. This could potentially lead to a situation where a process is mapped onto a CPU on one node, and assigned a GPU residing on another remote (potentially distant) node. This is undesirable for multiple reasons. First, the only visible GPU devices to a process are those that fall within the same node onto which the process has been mapped. Thus, accessing a remote GPU would require a middleware software framework (such as rCUDA [20]) to allow remote GPU virtualization. Second, all data movements between the process and its remote GPU would have to pass across the network. Third, some GPU-Aware MPI designs [21] leverage CPU-assisted mechanisms, in which the GPU-to-GPU communication is pipelined by staging the data chunks into the CPU memory. Such communications would highly suffer with remote GPU assignments.

A potential workaround for remote GPU assignment problem is to move (remap) the processes accordingly. That is, process mappings will be revised with respect to GPU assignments so that no process is assigned a remote GPU; each process will be moved to the same node as its assigned GPU. However, if we simply move the processes with respect to GPU assignments, it might increase CPU communication costs by placing highly communicating processes far from each other. Thus, it is required to consider CPU-to-CPU and GPU-to-GPU communications jointly to derive process mappings and GPU assignments in a unified framework.

V. MAGC Unified Topology-Aware Framework

A. *Unified Framework*

We propose the unified framework shown in Fig. 2 to tackle the topology-aware process mapping and GPU assignment problem shown in Fig. 1. The framework consists of three main phases that address the problem at different physical topology layers. The first phase mainly focuses on the inter-node network layer, whereas the second and third phases deal with the problem at the intra-node layer.

1) First phase: In order to avoid the problem of remote GPU assignment policy mentioned in Section IV, we limit GPU assignments to the boundaries of the node hosting each process. Thus, we first figure out the node on which each process should be mapped. However, we do this with respect to a combined communication pattern as the input to the mapping algorithm in the first phase. The combined pattern is built by adding the CPU and GPU communication patterns together. For the physical topology, we only need to consider the network topology at this phase because it will be the representative of the physical topology for both GPUs and CPUs at the inter-node layer of the system. In fact, this is the feature that enables us to use the combined CPU and GPU communication pattern in the first phase to conduct the mapping with a joint CPU-GPU awareness.

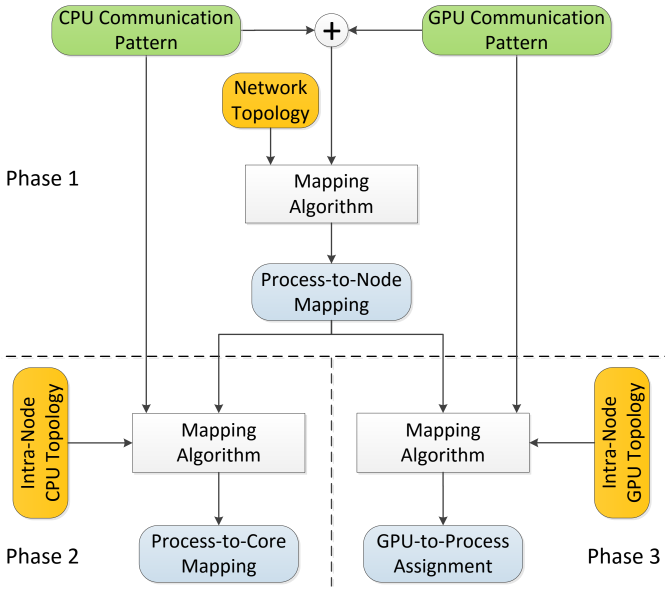


Figure 2: Schematic diagram of MAGC; three-phase approach for process mapping and GPU assignment.

The output of the first phase will be a process-to-node mapping. This mapping would also imply an implicit GPU assignment as it confines each process with the set of GPUs that reside on its hosting node. With an appropriate mapping algorithm, the mapping resulting from the first phase will have the following desired characteristics. The set of processes mapped on the same node will have a relatively higher combined CPU and GPU communications. This will potentially improve performance as such communications will benefit from stronger intra-node (versus inter-node network) communication channels. It will also decrease the potential need to have any remote GPU assignments in future steps. Furthermore, by using a network-aware mapping algorithm, we will gain similar benefits across the network as well; higher-communicating processes will fall into the nodes that are relatively closer to each other within the network.

2) Second phase: The second phase determines process-to-core bindings within each individual node. As shown by Fig. 2, at this phase, we only use the CPU communication pattern along with the intra-node CPU topology to calculate the desired mappings. The mapping result from the first phase is also fed to the algorithm to build the corresponding intra-node CPU communication pattern for each node. This is done by considering only the portion of the global CPU communication pattern matrix that corresponds to the set of processes mapped onto each node in Phase 1.

Note that there is no need to consider GPU-to-GPU communications at this stage as a process can be assigned any of the GPUs residing on its hosting node regardless of the core to which it is bound. In other words, we do not have the problem of remote GPUs within the boundaries of each single node. Moreover, at the intra-node level, CPU cores and GPUs do not necessarily have the same physical topology. Therefore, we leave GPU communications to the third phase where we attempt to optimize GPU assignments within each node based on the GPU communications and GPU physical topologies.

3) Third phase: In the third phase, we perform explicit assignment of GPUs to the processes within each node. At this stage, all processes have already been bound to a particular CPU core. As shown in Phase 3 of Fig. 2, GPU assignment is done with respect to the GPU communications of an application and the intra-node GPU topology. Again, the mapping result from the first phase is used to build the intra-node GPU communication pattern for each node. The explicit assignment at this phase is complimentary to the implicit GPU assignment that is enforced by the process mapping mentioned in the first phase. Here, the GPUs are assigned so as to further improve intra-node GPU communications within each node.

B. Mapping Algorithms

As shown in Fig. 2, MAGC uses a mapping algorithm to derive the output at each phase. In general, any mapping algorithm can be used for this purpose. The algorithms used at different phases could be the same or different from each other. By modeling the communication patterns and physical topologies in terms of graphs, one could use a general graph mapping algorithm/tool such as Scotch to do the mapping. Alternatively, one could use other fine-tuned mapping algorithms with respect to the particular physical topology of the system.

We will use the mapping approach proposed in [17] for the first phase of MAGC. For the specific systems used in our experiments in this work, such an approach would be simplified to its initial partitioning step because all nodes are connected to a single switch. For the second and third phases, we consider two design alternatives. In one case, we use Scotch as the mapping algorithm, whereas in another case, we use a greedy heuristic that attempts to directly optimize maximum congestion. We use the heuristic mainly to see if explicit modeling and optimization of congestion at the intra-node level can provide better results compared to Scotch. As the heuristic is used in both the second and third phase of MAGC, we would use the general phrase of PE rather than CPU core or GPU device in the following.

Congestion-based mapping heuristic: As a general graph mapping tool, Scotch models the physical topology by a weighted graph. This makes it possible to apply the library for various systems with different architectures. However, such generic graph model flattens the hardware architecture and does not represent the hierarchy in communication channels. Therefore, it cannot take congestion characteristics into account. Moreover, the specific edge weight values used for the graphs can highly affect the quality of Scotch results [22]. Finding an appropriate set of values can be non-trivial, specially as they can only be integers.

In our proposed heuristic, we use a tree to model the physical topology of the PEs. The tree will provide the actual hierarchy and structure of connections among the PEs. We use the HWLOC and NVML libraries to extract the topology of CPU cores and GPU devices within each node. Using the

Algorithm 1: Congestion-based greedy heuristic for intra-node mappings in Phase 2 and Phase 3 of MAGC

```
Input  : Set of processes $\mathcal{P}$, set of processing elements $\mathcal{PE}$,
         communication pattern matrix $\boldsymbol{C} = [c_{ij}]$, topology tree $T$
Output: The mapping $\tau : \mathcal{P} \rightarrow \mathcal{PE}$
1  $\mathcal{P}_M = \emptyset$; // Set of all mapped processes
2  $\mathcal{P}_M^c = \mathcal{P}$; // Set of all unmapped processes
3  $\mathcal{PE}_M = \emptyset$; // Set of occupied PEs
4  $\mathcal{PE}_M^c = \mathcal{PE}$; // Set of unoccupied PEs
5  initZero($T$); // Initialize link congestions to 0
6  $\boldsymbol{N} = [n_{ij}]_{|P| \times |P|}, n_{ij} = \begin{cases} 1 & c_{ij} > 0 \\ 0 & \text{otherwise} \end{cases}$;
7  while $\mathcal{P}_M^c \neq \emptyset$ do
8      for $q \in P_M^c$ do
9          $\delta = \frac{\sum_{r \in \mathcal{P}_M} (c_{qr} + c_{rq})}{\sum_{r \in \mathcal{P}_M} n_{qr}} - \frac{\sum_{s \in \mathcal{P}_M^c} (c_{qs} + c_{sq})}{\sum_{s \in \mathcal{P}_M^c} n_{qs}}$;
10     end
11     $p_{new} = q$ with the highest value of $\delta$;
12     for $pe \in \mathcal{PE}_M^c$ do
13         Temporarily assign $p_{new}$ onto $pe$ ;
14         calcCongestion($T$);
15         $maxCong$ = findMaxCongestion($T$) ;
16         $cost_{pe} = maxCong$ ;
17     end
18     $pe_{new} = pe_{min}$ ; // Choose PE with lowest cost
19     $\tau(p_{new}) = pe_{new}$ ; // Map new process to new PE
20     updateCongestion($T$) ; // Update the topology tree
21     $\mathcal{P}_M = \mathcal{P}_M \cup \{p_{new}\}$, $\mathcal{P}_M^c = \mathcal{P}_M^c \setminus \{p_{new}\}$;
22     $\mathcal{PE}_M = \mathcal{PE}_M \cup \{pe_{new}\}$, $\mathcal{PE}_M^c = \mathcal{PE}_M^c \setminus \{pe_{new}\}$;
23 end
```

topology tree, we keep track of the congestion imposed on each individual link of the physical topology, and use it as the main metric to figure out the desired mapping. To this end, we take into account the actual bandwidth of each link, and define congestion as the total traffic load that is passed through each link divided by its bandwidth.

Alg. 1 shows the heuristic details. It takes the set of processes, the set of PEs, communication pattern matrix, and the topology tree as input, and returns a mapping from the set of processes to the set of PEs. The algorithm iteratively chooses a new process and maps it to a desired PE in a greedy fashion. More specifically, at each iteration of the main loop in line 7, the new process for mapping is chosen based on the metric δ. The `for` loop in lines 8 to 10 calculates the value of δ for each unmapped process. The process with the highest value of δ is selected as the new process for mapping in line 11. As shown in line 9, δ captures the difference between two terms. For each unmapped process, these two terms respectively calculate the average of communication volume with the set of mapped and unmapped neighbors. Accordingly, matrix $\boldsymbol{N}$ in line 6 captures the neighborhood relationships among the processes. Such a metric δ allows us to choose a process whose communications are more associated with the set of mapped processes than the unmapped ones.

The new processing element for hosting the chosen process is selected throughout lines 12 to 18 of Alg. 1. In particular, the `for` loop in lines 12 to 17 evaluates each unoccupied PE for hosting the new process, and assigns a cost to it in line 16. The cost corresponding to each unoccupied PE is equal to the maximum congestion that will result from mapping the new process onto that PE. For each mapped neighbor n of p_{new}, we first find the links along the route from the PE that would potentially host p_{new} to the PE that hosts n. We then update the congestion of all such links with respect to the communication volume between p_{new} and n, which is given by the communication pattern matrix. Among all possible target PEs for p_{new}, we greedily choose the one that will lead to the lowest maximum congestion across all links within the physical topology tree (line 18). The new process is then mapped onto the new PE in line 19, and the topology tree is updated with the corresponding congestion values in line 20.

VI. Experimental Analysis

We conduct our experiments on two GPU clusters: Cluster A and Cluster B. Both clusters are equipped with multi-GPU nodes. Cluster A is composed of 4 nodes each having 16 K80 GPUs, two 12-core Intel Xeon processors operating at 2.7 GHz (total of 24 cores per node), and 256 GB of memory. Cluster B consists of 8 nodes each having 8 K20 GPUs, two 10-core Intel Xeon processors operating at 2.5 GHz (total of 20 cores per node), and 128 GB of memory. Both clusters run CentOS 6.7 and use Mellanox QDR InfiniBand as the interconnect. We use a total of 64 MPI processes in all our experiments with each MPI rank assigned to a single CPU core and GPU device. Also, we use Open MPI 1.10.2, CUDA 7.5, and Scotch 6.0.

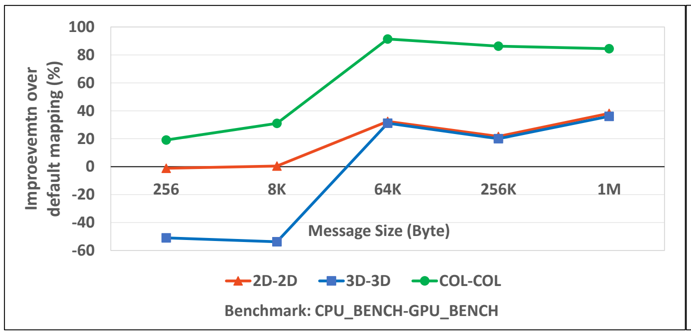
We evaluate the improvements achieved by MAGC over the default process mappings and GPU assignments. In addition to that, we also compare MAGC against two other straightforward strategies described below, so as to further evaluate the importance of considering GPU communications and using a unified framework.

a) Strategy 1: In the first strategy, we use the same three phases as MAGC, but ignore GPU communications in the first phase. Thus, the processes are 1) mapped to the nodes based on the network topology and the CPU communication pattern only, 2) locally bound to cores within each node based on the intra-node CPU topology and CPU communication pattern, and 3) locally assigned a GPU in each node based on the intra-node GPU topology and GPU communication pattern.

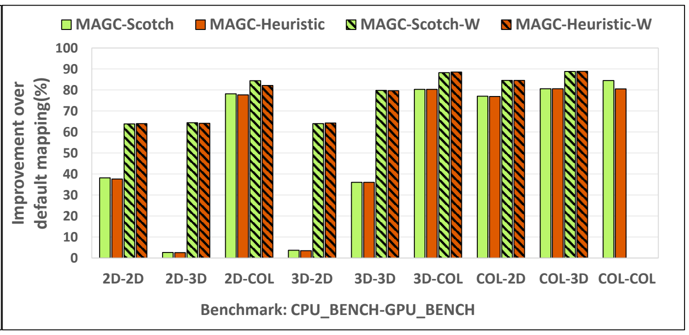
b) Strategy 2: The second strategy ignores the GPUs completely, and performs a traditional CPU-only process mapping. Thus, it only consists of two phases where the processes are 1) mapped to the nodes based on the network topology and the CPU communication pattern only, and 2) locally bound to specific cores within each node based on the intra-node CPU topology and CPU communication pattern.

A. Micro-benchmark Level Analysis

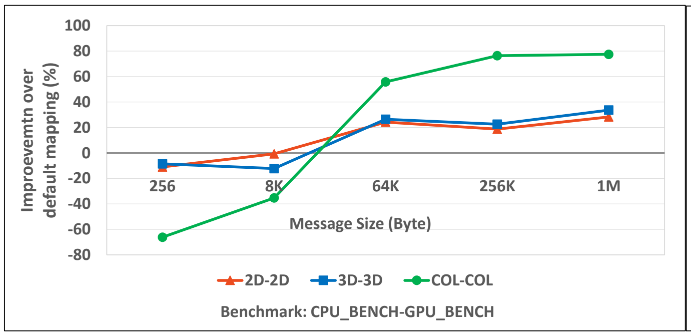
For our micro-benchmark analysis, we have developed a micro-benchmark suite that models various communication patterns among the CPU cores and among the GPU devices of a cluster. The current version consists of three main micro-benchmarks: 1) 2D_Stencil (2D), 2) 3D_Stencil (3D), and



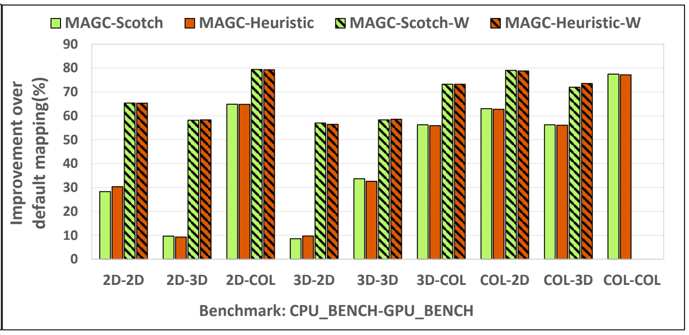
(a) Different message sizes with Scotch used in phases 2 and 3 - Cluster A



(b) 1MB message size and all micro-benchmarks - Cluster A



(c) Different message sizes with Scotch used in phases 2 and 3 - Cluster B



(d) 1MB message size for all micro-benchmarks - Cluster B

Figure 3: Micro-benchmark runtime improvements using MAGC on Cluster A and Cluster B (64 processes)

3) Sub-communicator collective (COL). The 2D and the 3D micro-benchmarks model a 2-dimensional 5-point and a 3-dimensional 7-point Stencil patterns respectively. The processes are organized into a 2D/3D mesh, and each process communicates with its two immediate neighbors along each dimension. For these two micro-benchmarks we consider two cases: a) *non-weighted* and b) *weighted*. In the former, we use the same message size for the communications along all dimensions, whereas in the latter, larger messages are used along the first dimension (3 times larger). In the sub-communicator collective micro-benchmark, the processes are organized into a 3-dimensional grid with a sub-communicator created for each group of processes falling along the first dimension. An MPI collective (MPI_Alltoall in our tests) is called over each sub-communicator.

Each of the micro-benchmarks can be independently used as the communication pattern among CPU cores and among GPU devices. We consider all possible combinations of such micro-benchmarks (9 in total) to model a wide variety of communication patterns. We represent each combination as an X-Y pair, where X and Y respectively denote the micro-benchmark of choice for CPU-to-CPU and GPU-to-GPU communications.

1) Improvements over the default mapping: Fig. 3 shows the corresponding results for Cluster A and Cluster B. It shows the communication time improvements achieved by using MAGC over the default process mappings and GPU assignments. In particular, Fig. 3(a) and Fig. 3(c) show the improvement trend across 5 different message sizes. For the sake of clarity and space, the results are only shown for three combinations of our micro-benchmarks, and Scotch as the mapping algorithm at the intra-node layer. The trend is similar for the cases not shown. It can be seen that for both clusters, improvements increase with message size. This is an expected behavior as MAGC is targeted for message volume and bandwidth optimizations, whereas smaller massages are more sensitive to message counts and startup latencies. In fact, we can see that for message sizes below 64KB, in some cases MAGC could result in performance degradation. However, for 64KB message size and above, MAGC can consistently improve performance for all micro-benchmarks. For the COL-COL case in particular, we see more than 80% improvement.

Fig. 3(b) and Fig. 3(d) show the improvements with 1MB message size for all micro-benchmarks. The figures show the results with both Scotch and the congestion-based heuristic as the mapping algorithms used in the second and third phases of MAGC. ‘MAGC-Scotch-W’ and ‘MAGC-Heuristic-W’ refer to the weighted versions of the 2D and 3D micro-benchmarks. As shown, on both clusters, MAGC can successfully improve performance for all micro-benchmark combinations. The highest improvements are achieved in cases that involve the sub-communicator all-to-all benchmark. For the COL-COL micro-benchmark, we can achieve 91.4% and 79.4% improvement on Cluster A and Cluster B respectively. Also, improvements are generally higher for the weighted versions of micro-benchmarks. This is because the larger messages communicated along one of the grid dimensions in the weighted micro-

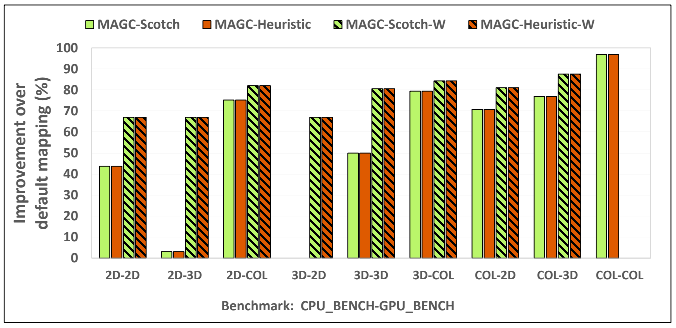


Figure 4: Maximum congestion improvements achieved by MAGC with Scotch and heuristic on Cluster A (64 processes)

benchmarks provide more room for optimizations. While the default mapping and GPU assignment is oblivious to the communication volume and bandwidth among different processes, our design takes advantage of such information, and improves performance by mapping intense communications on higher-bandwidth channels. The weighted micro-benchmarks also have a relatively less symmetric communication pattern, which again increases the chance for optimizations.

Another observation is that MAGC results in a similar performance with either Scotch or the heuristic. To understand the reason, we have evaluated both algorithms in terms of their impact on maximum congestion. Fig. 4 shows maximum congestion improvements achieved by MAGC with Scotch and the heuristic for all micro-benchmarks on Cluster A. The results show the combined improvements with respect to both CPU and GPU communications. The trend is similar for Cluster B and hence, we refrain from repeating it here. First observation is that in almost all cases, MAGC can successfully improve (decrease) maximum congestion. Second, Scotch and the heuristic have led to the same amount of improvement in maximum congestion for all cases.

The reason for such similar performance is two fold. First, MAGC uses the same fixed mapping algorithm in the first phase, regardless of the choice (Scotch or heuristic) for the second and third phases. Thus, the impact scope of the mapping algorithms used at the second and third phases is rather limited and bound to the intra-node layer. Second, the problem size at the intra-node layer is relatively small (16 for Cluster A, 8 for Cluster B) for which both Scotch and the heuristic result in a same performance. The differences between the two can be better studied on systems with higher numbers of CPU cores and GPU devices per node. Finally, it is worth noting that a good correlation is seen between the results in Fig. 4 for maximum congestion and those in Fig. 3 for runtime improvements.

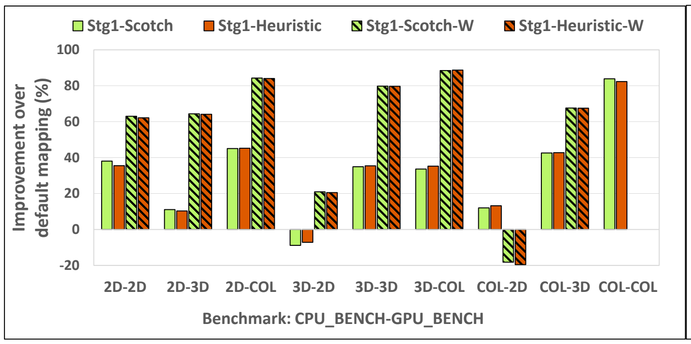
2) Comparison with straightforward strategies: Fig. 5 shows the improvements achieved over the default mapping with Strategy 1 and Strategy 2. Due to lack of space, we only present the results for Cluster A; the trend is similar for Cluster B. By comparing the results shown in Fig. 5 to those shown in Fig. 3(b), we can see that MAGC achieves higher performance improvements than Strategy 1 and Strategy 2 for most of the micro-benchmarks. In some cases (3D-2D non-weighted and COL-2D weighted), Strategy 1 and Strategy 2 even cause up to 20% degradation in performance, whereas MAGC can provide more than 80% improvement.

We also see very similar results for the two strategies in Fig. 5(a) and Fig. 5(b), implying that the third phase of Strategy 1 (i.e., local GPU assignment within each node) does not have major impacts on the performance. This, along with the better performance achieved by MAGC, shows the importance of considering GPU communications from the very first phase of the mapping methodology for GPU clusters.

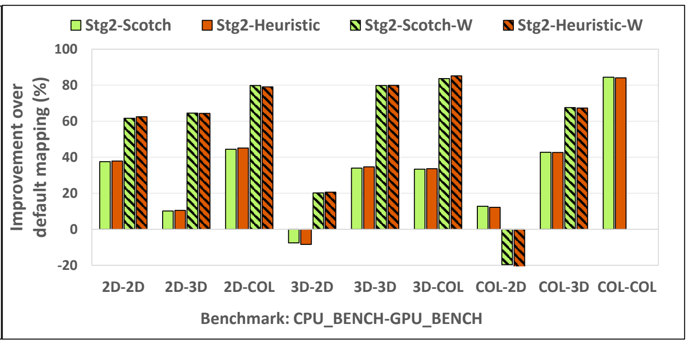
Another observation is the lower consistency in performance improvements provided by Strategy 1 and Strategy 2 compared to MAGC. We see more stable improvements for MAGC across different micro-benchmarks that is not affected by the type of the processing element (CPU or GPU) that runs each micro-benchmark. This is very well shown by the results for the 2D-COL versus COL-2D micro-benchmarks. While MAGC provides a consistent improvement (about 80%) in both cases, the two other strategies lead to very different results. MAGC achieves such consistency due to its joint consideration of both CPU and GPU communications.

A closer look at Fig. 5 reveals that Strategy 1 and Strategy 2 can achieve a similar improvement to MAGC in cases where the same micro-benchmark is used on both the CPUs and the GPUs (e.g., 2D-2D). This is expected because in such cases, optimizing for the CPU communications will also implicitly optimize the GPU communications as the two use the exact same pattern. When different micro-benchmarks are used for the CPU and GPU, Strategy 1 and Strategy 2 can perform comparable to MAGC only when we have a *weighted micro-benchmark on the CPU* accompanied by a *non-weighted micro-benchmark on the GPU*. For instance consider the 2D-COL case. We see more than 80% improvement with MAGC and the other two strategies. This is because in such cases, larger messages are exchanged among the CPUs compared to GPUs, which makes CPU communications the dominant factor in performance. However, when this is not the case (e.g., 3D-2D, COL-2D, COL-3D), we can clearly see MAGC's superior performance compared to the other two strategies.

3) Overhead analysis: We also evaluate MAGC in terms of the overheads. Fig. 6 shows the total time spent by MAGC with Scotch (MS) and the heuristic (MH). It also shows the time breakdown across each of the three phases. For Phase 2 and Phase 3, the results represent the aggregate time across all nodes. For the sake of space and clarity, we only present the results for three of the micro-benchmarks. The trend is very similar for other micro-benchmarks as well. We can see from the figure that MAGC imposes a low overhead which is less than 8 ms in all cases. We can also see that the overheads are lower with the heuristic compared to Scotch. This is specially true for Cluster B where we see considerably lower overheads with the heuristic at Phase 2 and Phase 3. This is mainly due to the fact that there are fewer number of CPU cores and GPU devices on Cluster B compared to Cluster A. Consequently, the



(a) Strategy 1 - 1 MB message size



(b) Strategy 2 - 1 MB message size

Figure 5: Micro-benchmark runtime improvements achieved by Strategy 1 and Strategy 2 on Cluster A (64 processes)

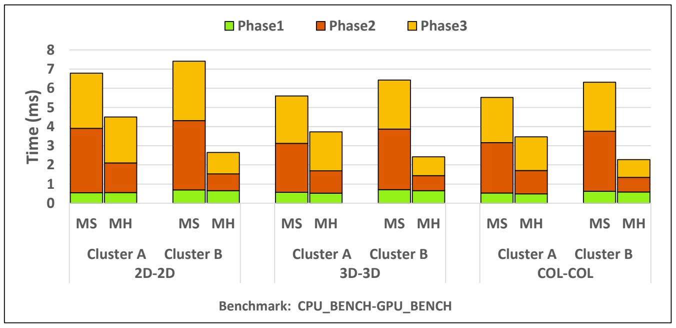


Figure 6: MAGC's overheads with Scotch (MS) and heuristic (MH) on Cluster A and Cluster B (64 processes)

heuristic will be dealing with a smaller topology tree for which the overheads of various steps in Alg. 1 (such as updating congestion values across the tree) will be lower. For Scotch however, the overheads are almost the same on both clusters and we see a bit of higher overhead for Phase 2 on Cluster B.

B. Application Level Analysis

In this section, we evaluate MAGC against a real application. To this end, we use HOOMD-blue which is a general-purpose particle simulation toolkit shown to scale from a single CPU core to thousands of GPUs. We run the application with 64 processes on Cluster A and Cluster B, with and without MAGC. More specifically, we consider two cases of single- and double-precision formats of the application. For the input, we use the classic Lennard-Jones (LJ) liquid benchmark with two different particle sizes: 512 thousand and 2 million.

1) Improvements over the default mapping: Fig. 7 shows the corresponding results in terms of the TPS (number of application time steps per second) improvements achieved from using MAGC in comparison with a default run of the application. We can see up to 6.5% and 8% improvements for Cluster A and Cluster B respectively. Moreover, for both clusters, the highest improvements are achieved with 512K particle size. For Cluster A, the improvements are higher for the single-precision case, whereas on Cluster B, we see the highest improvement for the double-precision case. We also see degradations in improvement with increase in particle size. We expected to see an opposite trend as larger particle sizes would increase the total volume of messages exchanged among processes. However, a larger particle size would also increase the total computation load of the application which could mask the impacts of communication improvements.

Our profiling results for HOOMD-blue show that the majority of communicated messages fall below 32KB on our platforms and with the workloads in use. Therefore, the messages are not large enough to consistently make the application bandwidth-bounded. Also, the communication pattern resembles a non-weighted 3-dimensional stencil with *wraparounds*, which makes the pattern quite symmetric. These are the main reasons for which we do not see greater performance enhancements for HOOMD-blue as they limit the room for potential optimizations through topology-aware mappings. We expect to see higher improvements for applications that use larger messages and/or employ irregular communication patterns. For small messages, it might be better to use MAGC with a latency-based heuristic to consider the latency characteristics of the physical topology rather than congestion/bandwidth.

2) Comparison with straightforward strategies: Finally, Fig. 8 shows the application improvements achieved by Strategy 1 and Strategy 2. Due to lack of space, we only show the results for Cluster A. According to the figure, we see lower improvements than those achieved by MAGC in Fig. 7. This is specially true for Strategy 2 where we can even see performance degradation in some cases. We also see better results for Strategy 1 compared to Strategy 2 which could be due to local topology-aware GPU assignments in Strategy 1.

VII. Conclusions

We discussed the joint problem of process-to-core mapping and GPU-to-process assignment in GPU clusters. Accordingly, we proposed a unified mapping approach (MAGC) that takes into account both the CPU and GPU communication patterns of an application, as well as the CPU and GPU physical topologies of the system. Our experimental results

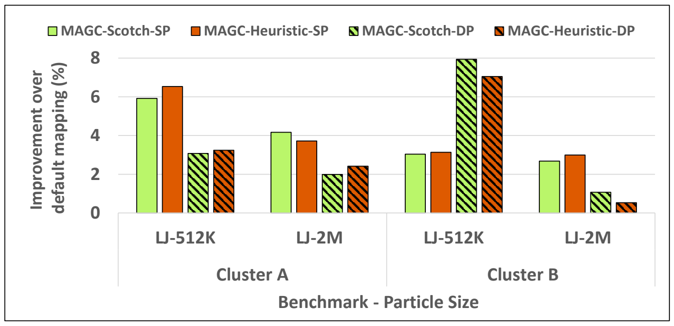


Figure 7: HOOMD-blue TPS (number of application time steps per second) improvements with MAGC (64 processes)

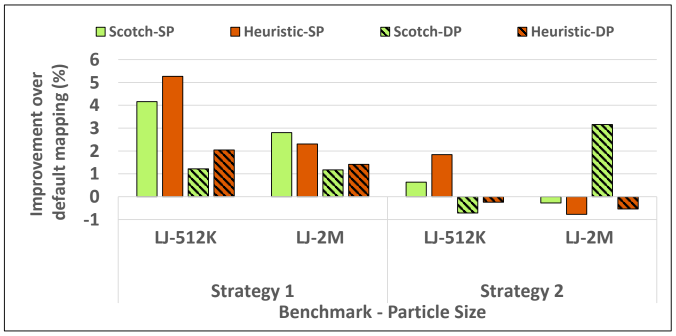


Figure 8: HOOMD-blue TPS improvements achieved by Strategy 1 and Strategy 2 on Cluster A (64 processes)

show that using MAGC, we can achieve considerable performance improvements by utilizing communication channels more efficiently. We also analyzed MAGC's performance with a congestion-based heuristic for intra-node mappings and showed that its performance is similar to Scotch, but with a lower overhead.

For future work, we intend to evaluate MAGC with other real applications. We are interested to see the impacts of MAGC on applications with irregular communication patterns and heavier communications. We also intend to evaluate MAGC when it is used with other mapping algorithms, and at larger scales. Incorporating virtualization mechanisms such as rCUDA to the framework is another venue for future work.

VIII. Acknowledgments

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant #RGPIN-2016-05389, Canada Foundation for Innovation and Ontario Innovation Trust Grant #7154. Computations were performed on the Helios supercomputer from Université Laval, managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), the ministére de l'Économie, de la science et de l'innovation du Québec (MESI) and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT). We especially thank Maxime Boissonneault for his technical support regarding our experiments.

References

[1] A. Bhatelé and L. V. Kalé, "An evaluative study on the effect of contention on message latencies in large supercomputers," in *Proc. Int. Symp. Parallel & Distributed Processing (IPDPS)*, 2009, pp. 1–8.

[2] P. Balaji, R. Gupta, A. Vishnu, and P. Beckman, "Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems," *Computer Science-Research and Development*, vol. 26, no. 3-4, pp. 247–256, 2011.

[3] A. Bhatelé and L. V. Kalé, "Heuristic-based techniques for mapping irregular communication graphs to mesh topologies," in *Proc. Int. Conf. High Performance Computing and Communications*, 2011, pp. 765–771.

[4] E. Jeannot and G. Mercier, "Near-optimal placement of MPI processes on hierarchical NUMA architectures," in *Proc. Int. Euro-Par Conf. Parallel Processing*, 2010, pp. 199–210.

[5] I. Faraji, S. Mirsadeghi, and A. Afsahi, "Topology-aware GPU selection on multi-GPU nodes," in *Proc. Int. Workshop on Accelerators and Hybrid Exascale Systems (IPDPSW/AsHES)*, 2016, pp. 712–720.

[6] F. Pellegrini and J. Roman, "Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs," in *Proc. Int. Conf. High-Performance Computing and Networking (HPCN)*, 1996, pp. 493–498.

[7] J. Glaser, T. D. Nguyen, J. A. Anderson, P. Lui, F. Spiga, J. A. Millan, D. C. Morse, and S. C. Glotzer, "Strong scaling of general-purpose molecular dynamics simulations on GPUs," *J. Comput. Phys.*, vol. 192, pp. 97–107, 2015.

[8] "MPI-3.1, http://www.mpi-forum.org/docs/mpi-3.1/ (last accessed 06/20/2016)."

[9] "MVAPICH2, http://mvapich.cse.ohio-state.edu (last accessed 06/20/2016)."

[10] "Open MPI, http://www.open-mpi.org/ (last accessed 06/20/2016)."

[11] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[12] T. Islam, K. Mohror, and M. Schulz, "Exploring the capabilities of the new MPI_T interface," in *Proc. European MPI Users' Group Meeting (EuroMPI/ASIA)*, 2014, pp. 91:91–91:96.

[13] H. Subramoni, S. Potluri, K. C. Kandalla, B. Barth, J. Vienne, J. Keasler, K. Tomko, K. Schulz, A. Moody, and D. K. Panda, "Design of a scalable InfiniBand topology service to enable network-topology-aware placement of processes," in *Proc. SC*, 2012, pp. 70:1–70:12.

[14] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "HWLOC: A generic framework for managing hardware affinities in HPC applications," in *Proc. Euromicro Int. Conf. Parallel, Distributed and Network-Based Processing*, 2010, pp. 180–186.

[15] "NVIDIA management library, https://developer.nvidia.com/nvidia-management-library-nvml (last accessed 06/20/2016)."

[16] T. Hoefler and M. Snir, "Generic topology mapping strategies for large-scale parallel architectures," in *Proc. Int. Conf. Supercomputing*, 2011, pp. 75–84.

[17] S. H. Mirsadeghi and A. Afsahi, "PTRAM: A parallel topology- and routing-aware mapping framework for large-scale HPC systems," in *Proc. Int. Workshop on High-Level Parallel Programming Models and Supportive Environments (IPDPSW/HIPS)*, 2016, pp. 386–396.

[18] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware MPI topology functions," in *Proc. EuroMPI*, 2011, pp. 50–60.

[19] M. Martinasso, G. Kwasniewski, S. Alam, T. Schulthess, and T. Hoefler, "A PCIe Congestion-Aware Performance Model for Densely Populated Accelerator Servers," 2016, to be published in Int. Conf. for High Performance Computing, Networking, Storage and Analysis (SC'16).

[20] J. Duato, A. J. Pena, F. Silla, R. Mayo, and E. S. Quintana-Ortí, "rCUDA: Reducing the number of GPU-based accelerators in high performance clusters," in *Proc. Int. Conf. on High Performance Computing and Simulation (HPCS)*, 2010, pp. 224–231.

[21] A. K. Singh, S. Potluri, H. Wang, K. Kandalla, S. Sur, and D. K. Panda, "MPI alltoall personalized exchange on GPGPU clusters: Design alternatives and benefit," in *Proc. Int. Conf. Cluster Computing*, 2011, pp. 420–427.

[22] E. Jeannot, G. Mercier, and F. Tessier, "Process placement in multicore clusters: Algorithmic issues and practical techniques," *IEEE Tran. Parallel and Distributed Systems*, vol. PP, no. 99, pp. 1–1, 2013.