Parallel Computing 68 (2017) 3–16

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Exploiting heterogeneity of communication channels for efficient GPU selection on multi-GPU nodes

Iman Faraji*, Seyed H. Mirsadeghi, Ahmad Afsahi

Department of Electrical and Computer Engineering, Queen's University Kingston, ON, K7L 3N6, Canada

ARTICLE INFO

Article history:
Received 15 October 2016
Revised 28 June 2017
Accepted 4 July 2017
Available online 8 July 2017

Keywords:
Multi-GPU node
MPI
Topology-aware GPU Selection
Mapping

ABSTRACT

Multi-GPU nodes have become the platform of choice for scientific applications. In a multi-GPU node, GPUs are interconnected together via different communication channels. The intranode communications among GPUs may traverse different paths with different latency and bandwidth characteristics. As the number of GPUs within a multi-GPU node increases, the physical topology of the GPU interconnects tend to have more levels of hierarchy, which in turn increases the heterogeneity of the GPU communication channels.

In this paper, we show that the performance of different intranode GPU communication channels can be considerably different from each other. Accordingly, we propose a topology-aware GPU selection scheme for efficient assignment of GPUs to the MPI processes within a node. The resulting assignment helps to improve the communication performance by mapping more intensive inter-process GPU-to-GPU communications on the stronger communication channels. We leverage three metrics in our scheme to distinguish among different GPU-to-GPU communication channels: latency, bandwidth, and distance. We evaluate our scheme through extensive experiments conducted on a 16-GPU node, and show that our scheme can provide considerable performance improvements over the default GPU selection scheme. In particular, we can achieve up to 70% and 21% performance improvement at the microbenchmark and application level, respectively.

1. Introduction

GPU accelerators have established themselves in modern HPC clusters. The high compute capacity and the lower power dissipation provided by them is what the HPC systems are striving for. In fact, GPUs have been deployed in many supercomputers in the Top500 [1]. Multi-GPU nodes are increasingly gaining popularity among the state-of-the-art HPC clusters. Such nodes have shown to be a promising platform for accelerating applications in different domains such as computational fluid dynamics [2], molecular dynamics [3–5], and weather forecasting models [6]. The architecture of these nodes usually consists of multi-core processors and multiple GPU devices interconnected by a hierarchy of different communication channels.

The Message Passing Interface (MPI) [7] is the de-facto standard for parallel programming. In GPU clusters, while processes can offload and accelerate computation on the GPUs, they require support from the MPI library to communicate their data that resides on the GPU memory. To perform such communications efficiently, MPI libraries should be tuned and become GPU-aware. In MPI applications, both the intranode and internode GPU-to-GPU communications are a major

* Corresponding author.
E-mail addresses: i.faraji@queensu.ca, imfaraji@gmail.com (I. Faraji).

http://dx.doi.org/10.1016/j.parco.2017.07.001
0167-8191/

performance bottleneck. In fact, the overhead imposed by the GPU communications[1] in applications with high data inter-dependency may even wipe out the potential benefits of GPU offloading. In this regard, various researchers have attempted to address the GPU communications bottleneck by incorporating GPU awareness into the MPI point-to-point and collective operations [8–10].

MPI processes running on a multi-GPU node can select any of the intranode GPU devices. However, depending on the selected GPU, inter-process GPU communications will traverse different paths within the physical topology of the intranode GPU interconnect. The traversal path may consist of different communication channels such as a single or multiple PCIe internal switches, a PCIe root complex[2], or an inter-socket interconnect such as Intel QPI. We show that in a multi-GPU node, not only should one expect heterogeneity in terms of the computational power and characteristics of different processing units (i.e., CPUs and GPUs), but also an added heterogeneity in terms of the topology and performance of different communication channels used to connect them. Taking this into account, our goal is to find an assignment of GPU devices to MPI processes that takes into account the heterogeneity of communication channels to utilize them more efficiently. This will in turn help to improve the performance of GPU communications.

In this paper, we first show the performance heterogeneity of various communication channels that connect the GPUs within a multi-GPU node. The results show a considerable variation in communication performance among different GPUs, specially for larger messages. Next, we propose a topology-aware GPU selection scheme in order to assign GPU devices to MPI processes based on the GPU communication pattern of the application, and the physical topology of the target multi-GPU node. We use three different metrics to model the physical topology characteristics: latency, bandwidth, and distance. We use these metrics to distinguish among different GPU-to-GPU communication paths within a multi-GPU node. We also profile the MPI application to extract its underlying GPU communication pattern. Next, we model the GPU assignment problem as a graph mapping problem, where we seek to map highly communicating pairs of processes to the GPU pairs that benefit from stronger physical connections.

We integrate our proposed scheme into the Open MPI library [11], and evaluate it through extensive experiments performed on a multi-GPU node consisting of 16 GPUs. To this end, we use three microbenchmarks and one real application. The microbenchmarks model three of the commonly used communication patterns in parallel applications. Namely, they model 2D and 3D stencil patterns, as well as alltoall communications over subcommunicators. For application evaluations, we use HOOMD-Blue [3,12] which is a general-purpose toolkit used for molecular dynamics simulations. The results show that our topology-aware GPU selection scheme can highly outperform the default selection scheme. At the microbenchmark level, we can achieve up to 70% performance improvement over the naive approach, with the performance referring to the total communication time. We also observe up to 21% performance improvement for HOOMD-Blue, with the performance referring to the application time steps per second.

The work in this paper extends our prior study in [13]. We expand our performance evaluations with an extended set of microbenchmarks and a comprehensive analysis of the results. We also provide further investigation of our microbenchmark results by: 1) analyzing the impact of our topology-aware schemes on the congestion across the GPU communication channels; and 2) comparing them against four random mappings. Moreover, we provide results for an end-point molecular dynamics application, namely HOOMD-Blue. We evaluate both the single- and double-precision versions of HOOMD-Blue with different input benchmarks and particle sizes. In order to provide a detailed analysis of the results, we also extend the FPMPI library [14] with support for GPU communications. We report the profiling results from our extended FPMPI for HOOMD-Blue to give more insights into the extent of improvements with our proposed topology-aware GPU selection scheme.

2. Background

2.1. MPI

The Message Passing Interface (MPI) is the de-facto standard for parallel programming. In MPI, communication is realized by explicit movement of data from the address space of one process to another. Accordingly, MPI provides support for various types of communications such as point-to-point, collective and one-sided. The GPU support has been added to the well-known implementations of MPI such as MVAPICH2 [15] and Open MPI [11]. The GPU support may follow a general approach which involves staging the GPU data into the host buffer and leveraging the CPU-based MPI routines. It may also involve further tunings by pipelining the transfers and using specifically designed algorithms for some MPI routines.

2.2. SCOTCH

In general, topology-aware mapping is an instance of the graph mapping (embedding) problem where a guest graph G is mapped onto a host graph H. The problem is known to be NP-hard, however, various mapping algorithms and heuristics exist that can provide sub-optimal solutions. Various graph partitioning libraries such as SCOTCH [16], METIS [17], and

[1] We interchangeably use 'GPU-to-GPU' and 'GPU' communications to refer to the communications among the GPU devices. Similarly, we interchangeably use 'CPU-to-CPU' and 'CPU' communications to refer to the communications among the CPU cores.

[2] Root complex is a part of the hostbridge; in the sequel, we use these terms interchangeably.

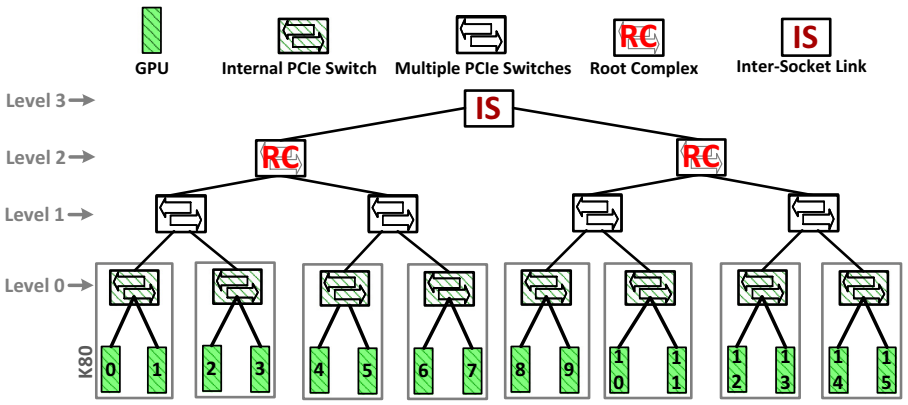


Fig. 1. Different intranode GPU pair levels.

ParaMETIS [18] have also been used by researchers for this purpose. In particular, SCOTCH provides a direct API for graph mapping. It is capable of mapping a given source graph onto a given target graph. These graphs may have any topology, and their vertices and edges can be weighted. In this work, we use the SCOTCH library to perform the mapping stage in our topology-aware GPU selection scheme.

2.3. NVIDIA Management Library

Information about the internal architecture of a node can be extracted by various tools. In particular, HWLOC [19,20] is a well-known library that provides a set of portable APIs to query the attributes of a node including cores, sockets, caches as well as I/O devices such as InfiniBand network interfaces [21] or GPUs. In this work, however, we only use the NVIDIA Management Library (NVML) [22] to extract the GPU topology of a multi-GPU node. The NVML library includes a set of C-based APIs that can be used for extracting various information about the NVIDIA GPU devices, including the topology information. We use the *nvmlDeviceGetTopologyCommonAncestor*() API in the NVML library to retrieve the common ancestor of each pair of GPUs. The retrieved common ancestor represents the highest level of hierarchy that the communication path between two GPUs will pass through.

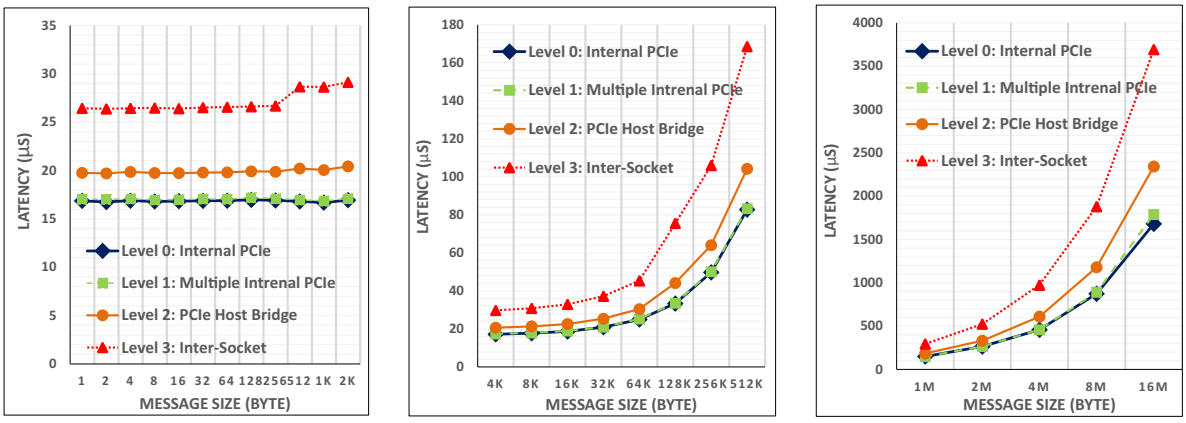
3. Motivation

In a multi-GPU node, various traversal paths may exist among different GPUs. Therefore, communication between different GPU pairs may go through different communication channels. Fig. 1 depicts an example configuration of a multi-GPU node which we also use as our experimental testbed (Section 5). According to the figure, GPUs can communicate with each other through different paths. In general, communication path between different GPU pairs can traverse four topology levels which we refer to as **Level 0, Level 1, Level 2**, and **Level 3**.

At **Level 0**, communication path between GPU pairs traverses a PCIe internal switch; this path exists among the on-board GPUs of a single GPU accelerator (e.g., path between **GPU 0** and **GPU 1**). Communication path at **Level 1** goes through multiple PCIe switches (e.g., path between **GPU 0** and **GPU 2**). At **Level 2**, communication path crosses a root complex (*RC*) device (e.g., path between **GPU 0** and **GPU 4**); *RC* connects the PCIe switch fabric to the socket. Communications at **Level 3** goes through an inter-socket (*IS*) link such as Intel QPI (e.g., path between **GPU 0** and **GPU 8**). Such a variety of communication channels at different topology levels can result in different GPU communication latency and bandwidth. To evaluate the latency and bandwidth characteristics of various intranode GPU topology levels, we perform intranode ping-pong latency and bi-directional bandwidth tests. We measure both the latency and bandwidth of communications for all possible GPU pairs in our 16-GPU node across various message sizes. Next, depending on the topology level, we categorize the latency and bandwidth results into four levels, and report the average values in Fig. 2 and Fig. 3, respectively. As shown by the figure, the GPU topology level has considerable impact on the GPU communication latency and bandwidth. We can also observe that as the message size increases, the latency/bandwidth gap between different topology levels becomes wider. We can also infer that by increasing the GPU pair topology level, the latency increases and the bandwidth drops.

4. Design and implementation

In this section, we propose a topology-aware GPU selection scheme in order to efficiently assign intranode GPUs to MPI processes so as to improve the intranode inter-process GPU communication performance. In our approach, we leverage the GPU communication pattern and the physical characteristics of the node, and model our topology-aware GPU selection scheme as a graph mapping problem where the GPU communication graph is mapped onto the GPU physical topology graph. A given solution of this mapping problem would designate a specific assignment of GPUs to MPI processes.



(a) Small messages (b) Medium messages (c) Large messages

Fig. 2. Impact of intranode GPU topology level on ping-pong latency.

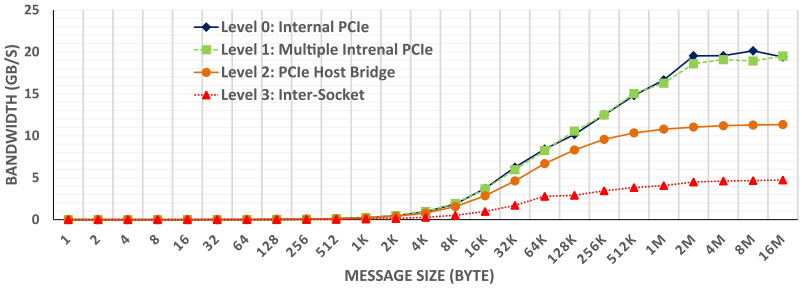


Fig. 3. Impact of intranode GPU topology level on bi-directional bandwidth.

We extract the GPU communication pattern by profiling the application in an initial run. In this regard, we instrument the MPI library to gather the GPU inter-process communications, and save into a square matrix. The matrix captures the total volume of GPU messages transferred between each pair of processes. Using this matrix, we construct a GPU *virtual* topology graph, representing the application's GPU communication pattern. In this graph, vertices stand for MPI processes, and the weighted edges represent the existence and significance of GPU communications among each pair of processes. Thus, the higher the edge weight, the higher the communication volume between the associated GPU peers.

We use three different metrics to reflect the impact of different topology levels in a multi-GPU node: 1) latency; 2) bandwidth; and 3) distance. For each metric, we perform a series of tests to extract the associated physical topology matrix file (note that none of the tests require root access). Using the generated files, we construct the GPU physical topology graph. Vertices in this graph represent the GPU device indices, and the edges represent the strength of the connection between two GPUs. All intranode GPUs are capable of communicating with each other, thus the GPU physical topology graph will be a complete graph. A higher edge value represents a lower latency, higher bandwidth, and lower communication distance in the latency-based, bandwidth-based, and distance-based physical topology graphs, respectively.

We use the SCOTCH library to map the constructed virtual topology graph onto the physical topology graph. We also use this library to construct the GPU virtual and physical topology graphs out of the virtual and physical matrix files. These matrix files are required to be available prior to the application execution. The virtual topology matrix file is generated once in an initial profiling run. The physical topology matrix file is created for each target multi-GPU node.

Depending on the chosen metric, we use different tests to generate the physical matrix file. For the latency-based metric, we use the normalized GPU pair latency values from the ping-pong test results (Fig. 2). For each pair of GPUs, we calculate the ratio of the Level 3 communication latency to the latency of the topology level associated with each pair of GPUs. To this end, we consider all latency values for message sizes in the range 1B to 2KB, and store the resulting average in the topology matrix file.

For the bandwidth-based metric, we use the normalized GPU pair bandwidth values from the bi-directional bandwidth test results (Fig. 3). For each GPU pair, we first calculate the ratio of the bandwidth corresponding to the topology level of each GPU pair to the bandwidth of the Level 3. To this end, we only consider the message sizes with varying bandwidth values (2KB to 16MB). For each GPU pair, we store the average ratio in the physical topology matrix file.

Finally, for the distance-based metric, we use a set of APIs from the NVML library to extract the communication distance of different GPU pairs that are available on the node. In this regard, we mainly use the NVML topology APIs such as *nvmlDeviceGetTopologyCommonAncestor*() with which we retrieve the common ancestor for all GPU pairs. For each pair of GPUs, the depth of their common ancestor can represent the physical distance between them. Based on the maximum number of detected topology levels (4 in our case), for each GPU pair, we store the difference between this maximum value and the topology level value of the pair into the physical topology matrix file.

We have integrated our proposed topology-aware GPU selection scheme into the MPI initialization phase of the Open MPI library. During this phase, we use the SCOTCH library to map the GPU virtual topology graph onto the physical topology graph. The output mapping table determines the desired GPU-to-process assignments in terms of an array M. The element $M[i]$ designates the GPU_id to be assigned to rank i. Due to the nature of the SCOTCH mapping algorithms, which can lead to different mapping results in different runs, only one process[3] performs the mapping and scatters the results M to other processes. Upon receiving the GPU_id, each MPI process calls the CUDA device selection function *cudaSetDevice*(GPU_id) to select its associated GPU.

5. Results and discussion

Our experiments are conducted on one node of the 6-node K80 Helios supercomputer installed at Universitè Laval. Each K80 node is equipped with 16 GPUs, 256 GB of memory, and two Intel Xeon Ivy Bridge E5-2697 v2 processors. Each of the Xeon processors provide 12 cores, operating at 2.7 GHz clock speed. Thus, there exist a total of 24 cores per node. Moreover, each node runs a 64-bit CentOS 6.7 as the operating system, and we use Open MPI 1.10.2, CUDA 7.5, and SCOTCH 5.8. Finally, we run all our experiments with 16 MPI processes (one process per GPU) that are evenly distributed among the cores on the two sockets.

5.1. Microbenchmark studies

In this section, we compare our proposed topology-aware scheme against the conventional GPU selection scheme at the microbenchmark level. In the conventional scheme, each process selects a GPU based on its associated rank number. For instance, a common approach is for process with rank i to select the GPU with index i. In our proposed scheme however, each process selects its target GPU based on the results from the topology-aware GPU assignments. In accordance with the three different metrics used in our scheme to model the physical topology of GPUs, we consider the following three scenarios for each of our experiments: 1) latency-map; 2) bandwidth-map; and 3) distance-map.

In particular, we use the three microbenchmarks that model communication patterns commonly used in parallel applications: 1) 2D Stencil (*2D*); 2) 3D Stencil (*3D*); and 3) Sub-communicator collective (*COLL*). The *2D* and *3D* microbenchmarks represent the communication pattern used in many stencil codes. The processes are first organized into a logical 2-dimensional (3-dimensional) grid, and then each process communicates with its immediate neighbors along each of the dimensions of the grid. With 16 processes, the *2D* and *3D* microbenchmarks organize the processes into a 4×4 and a $2 \times 2 \times 4$ grid, respectively. In addition, we consider the options of having *wraparound* and *weighted* connections for these microbenchmarks. In this regard, the microbenchmarks with wraparound will include an extra link for connecting the processes at the two edges of each dimension together. Microbenchmarks without wraparound do not have any such links. On the other hand, in the weighted case, a higher weight is assigned to the communications along a specific dimension of the grid. In other words, each process sends and receives larger messages ($3\times$ larger) to its neighbors that fall along the first dimension of the grid. In the non-weighted case, the same message volume is communicated along all dimensions. Thus, there are four cases for the *2D* and *3D* microbenchmarks: 1) Without weight and without wraparound; 2) Without weight, but with wraparound; 3) With weight, but without wraparound; and 4) With weight and with wraparound.

In the sub-communicator collective microbenchmark (*COLL*), the processes are first organized into a 3-dimensional grid ($4 \times 2 \times 2$), and an MPI sub-communicator is created for each group of processes that fall along the first dimension of the grid. Next, an MPI_Alltoall is called over each sub-communicator.

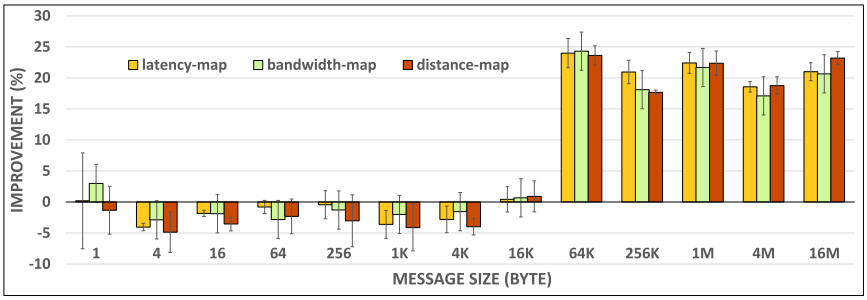
5.1.1. Performance results

Fig. 4 through Fig. 6 show the results in terms of the improvements in the communication time of each microbenchmark. We report the improvements achieved by our topology-aware scheme over the default (naive) GPU selection. In each case, we report the average of four runs and include standard deviation.

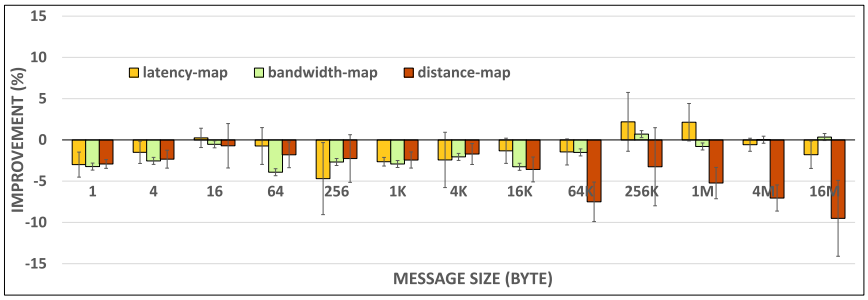
According to Fig. 4, for the non-weighted *2D* and *3D* microbenchmarks with no wraparound, performance improvement is achieved mainly for message sizes larger than 64KB. With wraparound connection, in the *2D* case (Fig. 4(b)), no performance improvement can be achieved for any of the metrics. We can also observe some performance degradation using the distance metric. For the *3D* case (Fig. 4(d)), all metrics provide the same improvement across all messages.

Fig. 5 shows that for the weighted *2D* and *3D* microbenchmarks, we can achieve up to 65% performance improvement by using the topology-aware GPU selection scheme. In addition, we can also see that the bandwidth metric consistently

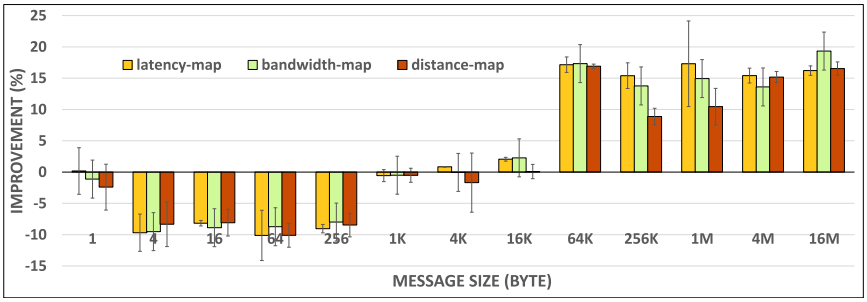
[3] without loss of generality, process with rank 0



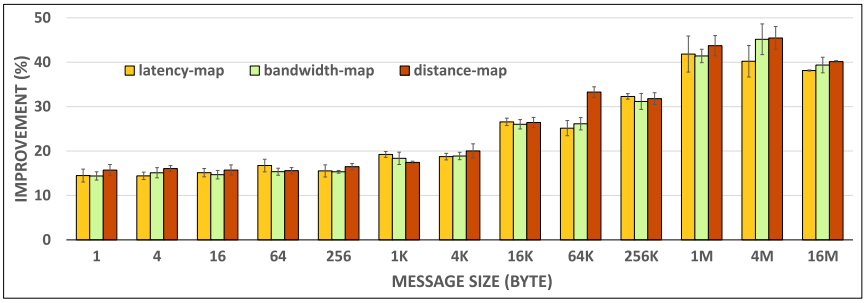
(a) 2D without wraparound without weight



(b) 2D with wraparound without weight

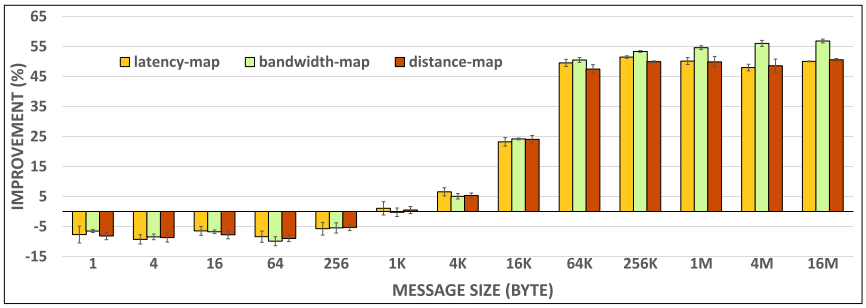


(c) 3D without wraparound without weight

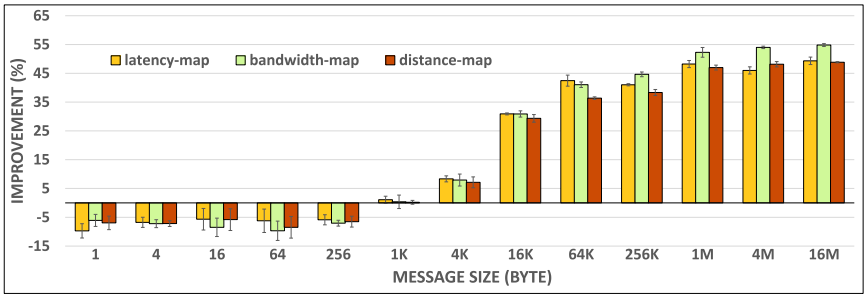


(d) 3D with wraparound without weight

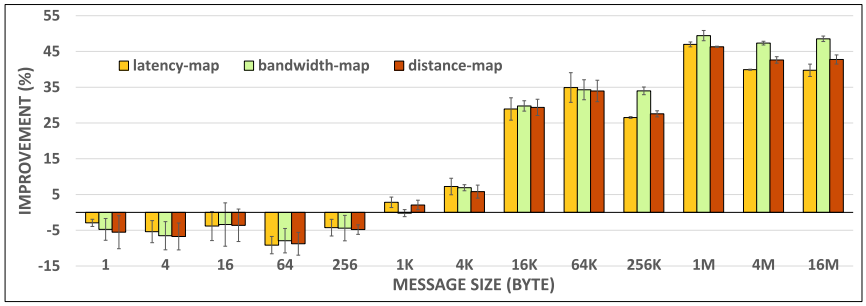
Fig. 4. Communication time improvements achieved by topology-aware GPU selection over the default selection scheme for the non-weighted 2D and 3D microbenchmarks.



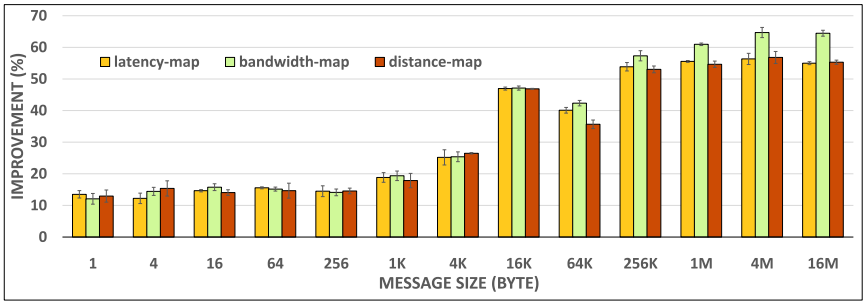
(a) 2D without wraparound with weight



(b) 2D with wraparound with weight



(c) 3D without wraparound with weight



(d) 3D with wraparound with weight

Fig. 5. Communication time improvements achieved by topology-aware GPU selection over the default selection scheme for the weighted 2D and 3D microbenchmarks.

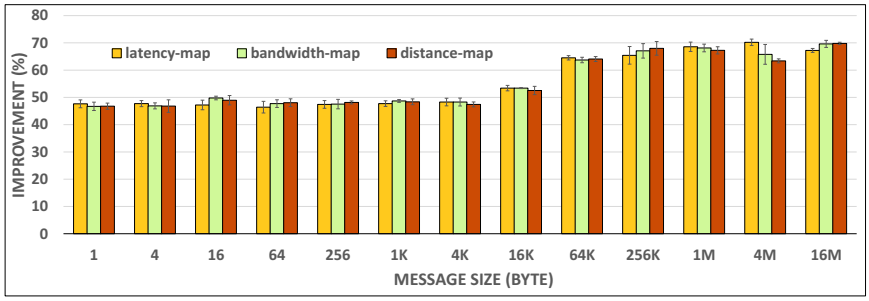


Fig. 6. Communication time improvements achieved by topology-aware GPU selection over the default selection scheme for the sub-communicator collective microbenchmark.

provides an equal or higher improvement compared to the latency and distance metrics. Fig. 6 shows that for the *COLL* microbenchmark, all three metrics can improve the performance by up to 70% across all messages.

We also observe that the weighted *2D* and *3D* microbenchmarks can benefit more from our topology-aware GPU selection scheme when compared to their non-weighted counterparts. This is an expected behavior because in the weighted cases, the GPUs should be assigned to the processes in a way that the heavier-communicating processes end up using the GPU pairs with stronger physical connections. This would in turn provide more opportunity for performance optimizations through topology awareness.

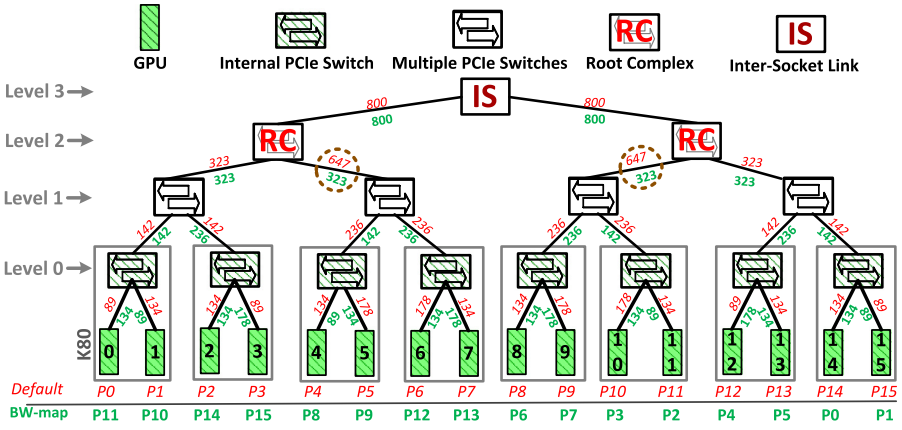
On the contrary, we do not see the same behavior in presence of the wraparound connections. In fact, while adding wraparound connections would lead to further improvements for the *3D* microbenchmark, we do not observe any improvements for the *2D* microbenchmark; we can even see some performance degradation. To investigate this further, we analyzed the communication pattern of these microbenchmark with and without the wraparound connections. We noticed that wraparound connections in the *3D* microbenchmark will result in communications between processes that are far from each other (in terms of the GPUs assigned to them by the default approach). As shown in Fig. 2 and Fig. 3, the farther two GPUs are from each other, the lower their communication performance would be. In the *2D* case on the other hand, adding wraparound connections will cause most of the communications to take place among the GPUs with stronger connections among them. In conclusion, adding wraparound connections makes the default GPU assignment an already-good match for the *2D* microbenchmark communication pattern, whereas it is the opposite for the *3D* microbenchmark.

In general, we can observe that our topology-aware schemes are more beneficial for large messages (greater than 16KB). The reason, as shown in Fig. 2 and Fig. 3, is that the difference in the latency of the small messages at different levels is not as much as the difference in the bandwidth of the large messages. More specifically, the latency ratio of **Level 1, Level 2**, and **Level 3** to **Level 0** for small messages is 1.01, 1.17, and 1.57, respectively. On the other hand, the bandwidth ratio of **Level 0** to **Level 1, Level 2**, and **Level 3** for large messages is 2.47, 4.22, and 4.47, respectively. So, the main difference between the channels at different levels is in their bandwidth. As a result, the performance of different topology levels are more diverse for large message sizes as they are mostly affected by the bandwidth characteristics of the underlying communication channels. For small messages, on the other hand, there is much less room for improvement (1.57 latency ratio compared to 4.47 bandwidth ratio). In addition, small-message communications are highly affected by the startup latencies which are not affected by topology awareness and the specific strategy used for GPU assignment. Taking this into consideration, there are still three microbenchmarks for which we can get performance improvement for small messages. There are also some cases where we see slight performance degradation for small messages. We leave further investigations on this trend for small messages to a future work. Designing a scheme that is finely tuned for small-message communications falls within the scope of our future work.

5.1.2. Congestion analysis

We further investigate our results by analyzing how our topology-aware scheme affects the congestion across the GPU communication channels. We define congestion as the total volume of the traffic that passes through a link divided by the bandwidth of that link. Different mappings will result in different congestion across the links that interconnect GPUs. We would like to find a mapping that leads to lower congestion across the communication channels.

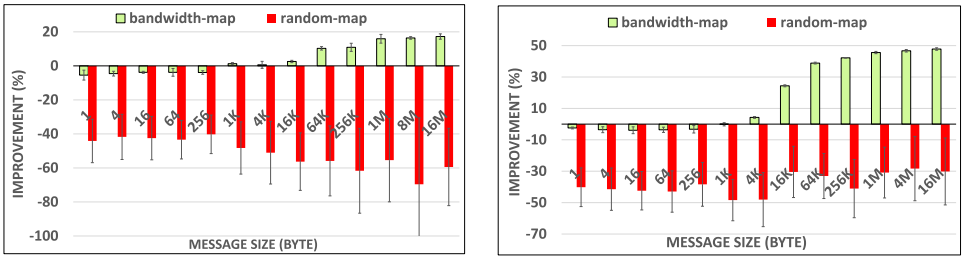
In the following, we compare our topology-aware scheme against the default mapping with respect to their resulting congestion for two of our microbenchmarks: 2D non-weighted and 3D weighted (both without wraparound). Fig. 7 shows the congestion values for the 2D and 3D microbenchmarks. The numbers above each link (shown in red) represent the congestion values for the default mapping, while the numbers below each link (shown in green) represent the congestion values for our topology-aware mapping using the bandwidth metric. Moreover, the numbers below each GPU denotes the rank of the process mapped onto that GPU by the default and topology-aware schemes, respectively. As shown in Fig. 7(a), for the 2D non-weighted benchmark the congestion values for both mappings are similar to each other except at **Level 2** (encircled area), where our topology-aware mapping leads to a lower congestion. On the other hand, as shown by Fig. 7(b), for the 3D weighted benchmark the topology-aware mapping can significantly decrease the congestion values across the links



(a) 2D without wraparound without weight

(b) 3D without wraparound with weight

Fig. 7. Congestion values for the default and topology-aware GPU assignment.



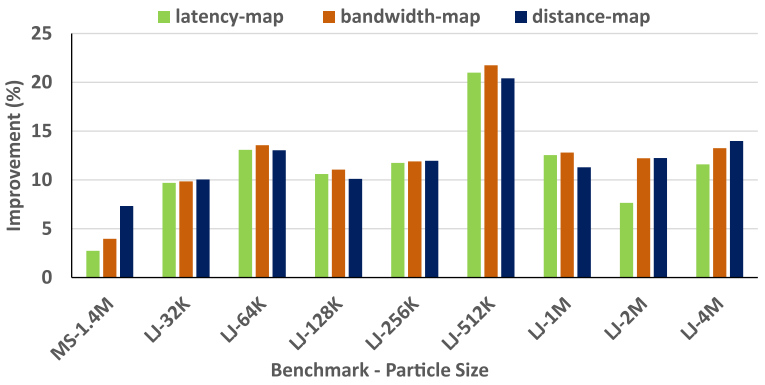
(a) 2D without wraparound without weight (b) 3D without wraparound with weight

Fig. 8. Communication time improvements achieved by topology-aware GPU selection and random mapping over the default selection scheme.

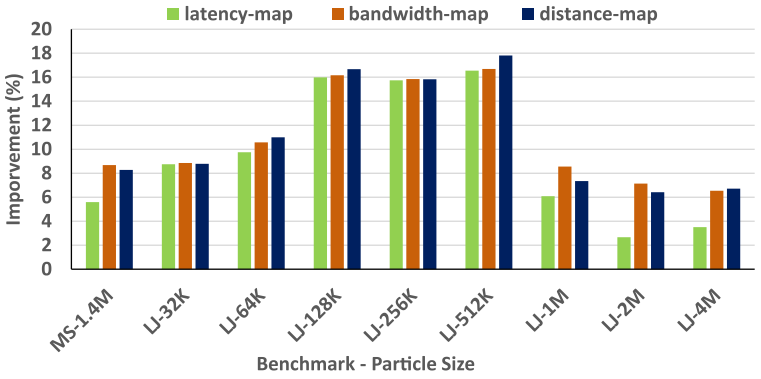
at different levels of the tree. In particular, the maximum congestion at **Level 3** is decreased from 1600 to 264 (encircled area). This also correlates with our microbenchmark performance results for which we achieve higher communication time improvements for the 3D weighted benchmark ($\approx$ 50% in Fig. 5(c)) compared to the 2D non-weighted ($\approx$ 20% in Fig. 4(a)).

5.1.3. Comparison with random mapping

To further show the benefits of the topology-aware GPU selection, we compare our approach (using bandwidth metric) against four different random mappings. These mappings use a random sequence of numbers that is generated for each run. We change the random seed across different runs to avoid getting the same results. We provide our results as the average of four runs (with standard deviation) in Fig. 8. According to the figure, the random mapping for both benchmarks underperforms the default mapping. We can also see a higher standard deviation for the random results compared to our scheme. Moreover, we did not observe any performance improvement with any of the four random mappings over the default mapping. This highlights the importance of using a non-trivial and topology-aware design for GPU assignment.



(a) Single-precision



(b) Double-precision

Fig. 9. TPS improvement of topology-aware mappings over default mapping on a) single-precision and b) double-precision HOOMD-blue Application.

5.2. Application study

In this section, we evaluate our proposed scheme across an end-point application: HOOMD-Blue [3,12]. HOOMD-Blue (Highly Optimized Object-oriented Many-particle Dynamics-blue edition) is a general-purpose toolkit used for molecular dynamics simulations. It supports multi-CPU and multi-GPU simulations using MPI, with a one-to-one mapping between CPU cores (MPI ranks) and GPU devices. Each GPU is assigned a sub-domain of the multi-dimensional simulation box. The length of the sub-domains are calculated by dividing the lengths of the box by the number of processors per dimension..

We configure HOOMD-Blue for both single- and double-precision computations, and use two different benchmarks: 1) MicroSphere (MS), and 2) Lennard-Jones (LJ) liquid. The MS benchmark simulates 1,428,364 particles; it runs a system of star polymers in an explicit solvent which organize into a microspherical droplet. The LJ benchmark is a classic benchmark in general-purpose molecular dynamics simulations. It is representative of the performance that HOOMD-Blue can achieve with straight pair potential simulations. We use a range of particle sizes from 32,000 to 4,000,000 with this benchmark.

5.2.1. Application performance results and analysis

Fig. 9 reports the TPS (number of application Time steps Per Second) improvements using our scheme over the default approach for the single- (Fig. 9(a)) and double-precision (Fig. 9(b)) versions of HOOMD-Blue. According to the figure, our scheme can provide up to 21.7% and 17.8% improvement for the single- and double-precision versions, respectively. We can also see that in most cases, the bandwidth and the distance metrics tend to outperform the latency metric. The general trend in Fig. 9 shows that the improvement increases as the particle size grows up to 512K and then drops. In order to understand the reason behind such a drop in improvement, we need to evaluate the communication characteristics of the application in detail. To this end, we use the FPMPI [14] library to get more insights into the nature of the communications in HOOMD-Blue. FPMPI [14] is a profiling library which provides various information about the underlying MPI communications of an application. However, FPMPI does not distinguish between the CPU and GPU communications. In this regard, we have extended the FPMPI library to provide profiling support for both CPU and GPU communications. The extended profiler allows us to separately extract the CPU and GPU communication characteristics of an application. To this end, we leverage various

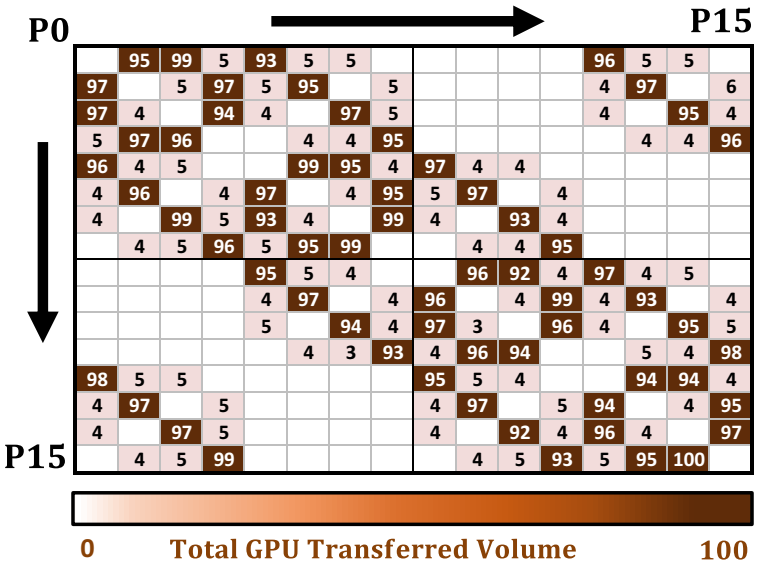


Fig. 10. Normalized GPU communication pattern of double-precision HOOMD-Blue with LJ-512K benchmark.

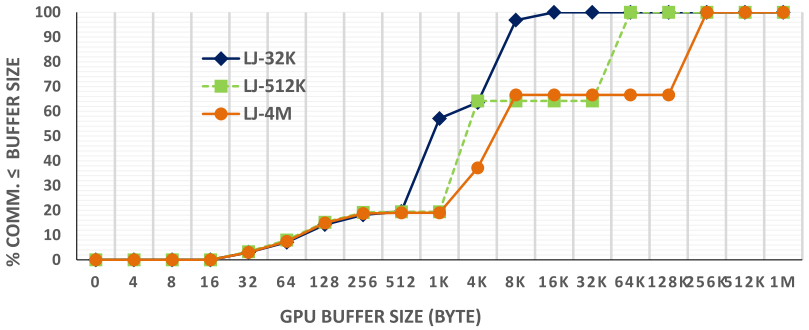


Fig. 11. Distribution of different message sizes in GPU communications of double-precision HOOMD-Blue with LJ benchmark.

CUDA APIs to analyze the buffer(s) in MPI routines. By analyzing the buffer(s), we can determine whether it is located on the host main memory or on the GPU global memory. We also instrument Open MPI to expose specific information that will be queried by the FPMPI library. For instance, we add the address type of the send/receive buffers of MPI routines to the MPI_Request object to distinguish among CPU and GPU communications.

Fig. 10 demonstrates the GPU communication pattern of the double-precision HOOMD-Blue with LJ-512K benchmark. The pattern is represented by a square matrix, where each row and column corresponds to an MPI rank. The *ij*th element of the matrix denotes the total volume of messages transferred between the pair of GPU devices assigned to rank i and j[4]. As shown by Fig. 10, only a small portion (around 20%) of the GPU pairs are involved in heavy communications. More importantly, the communication pattern closely resembles the pattern induced by a 3D Stencil with wrap-around and non-weighted connections. On the other hand, Fig. 11 shows the distribution of GPU communications across various message sizes (averaged over all ranks) for three different particle sizes. We can observe that as the particle size increases, the share of the larger message sizes also increases. With such communication characteristics and in accordance to the result shown in Fig. 4(d), we expect to observe a non-decreasing improvement with the increase in particle size for HOOMD-blue. However, the highest improvement is achieved with the 512K particle size in Fig. 9. To further investigate our application results, in Fig. 12 we provide the share of CPU and GPU communications in the total application runtime on three benchmarks with different particle sizes. According to this figure, our mapping schemes can improve the total application runtime for all three benchmarks. We can also observe that our mapping schemes only improve the GPU communication portion of the application and the rest of the application runtime is almost left intact. Fig. 12 also shows that as the particle size increases, the share of GPU communications in the total application runtime decreases. Consequently, any GPU communication improvements would have a relatively lower impact on the total application performance. To verify this, we also measure the *GPU*

[4] Note that the values have been normalized between 0 and 100.

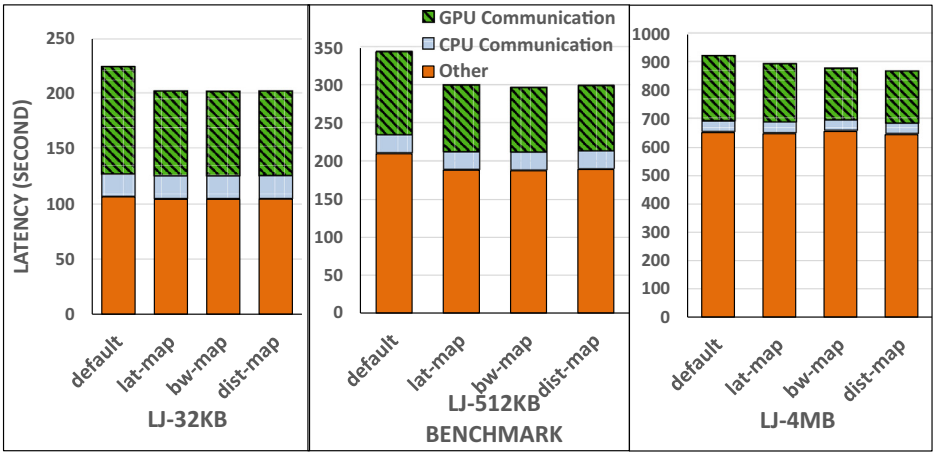


Fig. 12. Share of GPU Communications in total HOOMD-Blue runtime.

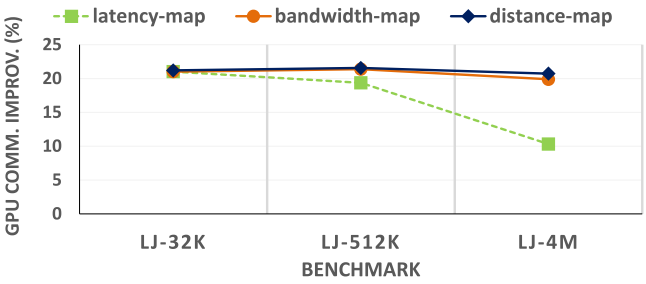


Fig. 13. HOOMD-Blue GPU communication improvements.

communication time improvements achieved by our scheme for the HOOMD-Blue application. Fig. 13 shows the corresponding results. This time, we can see a steady improvement of about 20% for our topology-aware scheme across all particle sizes. The only exception is with the latency metric and 4M particle size for which we see a reduction in the achieved improvement. This is expected to some extent, as latency is not a good representative of communication channel characteristics for large messages. In conclusion, although larger message sizes provide more opportunity to improve communication performance thorough our topology-aware scheme, the lower share of GPU communications in the total runtime of the application leads to a lower reflection of such improvements. This is why we see a drop in the achieved performance improvements after the 512k particle size shown by Fig. 9. However, we see a generally increasing trend in the improvements up to (and including) 512K particle size. This implies that the 512K particle size is a turning point in terms of the impact of *GPU message size* versus *communication time share* on the overall improvements.

5.2.2. Mapping overhead

In this section, we analyze the overhead of using our proposed topology-aware GPU assignment scheme. To this end, we measure the mapping time and compare it with the application runtime. According to our results, regardless of the mapping metric, benchmark type, and the particle size that are used for the application, the mapping time is around 0.5 ms. This time is negligible compared to the total application runtime and contributes to at most 0.0003% of that. This mapping time is also considered to be a one-time overhead for each instance of the application. We note that the main purpose of this paper is to show the importance of topology-aware GPU selection and the mapping choice falls into the scope of another study. Thus, one may replace SCOTCH with another mapping heuristic. In particular, for small multi-GPU nodes (with 4 or 8 GPUs), one could use an exhaustive search to find the optimal mapping.

6. Related work

In GPU clusters, GPU communications play a crucial role in the performance of MPI applications. In this regard, researchers have studied various GPU-aware point-to-point and collective operations to improve the GPU communication performance [8–10]. Faraji and Afsahi [8] propose an intranode GPU-aware MPI_Allreduce in which CUDA IPC is used to gather the pertinent data into the shared GPU buffer, followed by an in-GPU reduction. The CUDA IPC copy type is also utilized in [9,10] to improve one-sided and point-to-point communications. While these studies target to improve the efficiency of

specific MPI routines, our goal in this paper is to improve the total MPI communication among GPUs by efficiently assigning them to MPI processes.

Martinasso, et al. [23] provide a detailed analysis of the congestion behavior associated with the PCIe fabric that is used to connect the GPUs in a multi-GPU node. Accordingly, a congestion-aware performance model is proposed that can be used to predict the communication times in presence of congestion on a given PCIe topology. The proposed model can help to design more efficient algorithms for intra-node GPU communications. Lutz, et al. [24] propose an autotuning framework for distribution of stencil computations across multiple GPUs. They show that various PCIe layouts could have adverse effects on the performance, thereby utilizing all GPUs might not be necessarily a better choice in all cases. Similarly, we show that different communication channels among GPUs can have different performance characteristics, however we exploit this to propose a topology-aware GPU assignment for MPI processes.

Topology-aware mapping has been extensively studied in the context of CPU communications. Several experiments performed on large-scale supercomputers have verified the adverse effects of contention and hop-count on message latencies [25]. Another study [26] shows that different mappings of an application on large-scale IBM BG/P systems can significantly affect the overall performance. Rashti, et al. [27] propose a topology-aware mapping mechanism for two of the MPI topology functions. Mércier and Jeannot [28] modify the distributed graph topology function in MPICH2 to provide it with a topology-aware reordering of processes. Mirsadeghi, et al. [29] propose a parallel mapping approach that considers the underlying routing mechanism in addition to topology. Unlike these work, we exploit topology awareness and mapping concepts in the context of GPU communications, and show how a topology-aware GPU selection scheme can improve the GPU communication performance on a multi-GPU node.

7. Conclusion and future work

In this paper, we showed that intranode GPU topology can have significant impact on the communication performance. Accordingly, we proposed a non-trivial topology-aware GPU selection scheme that considers the application communication pattern and the physical topology of the multi-GPU node. We modeled the problem as a graph mapping problem and used the SCOTCH library to solve it. Our experimental results show that our proposed scheme can highly improve the communication performance at both the microbenchmark and application levels. Microbenchmarks with more distant communications between the GPUs (with respect to their GPU ID) are more susceptible to improvement by our topology-aware scheme. Also, more improvement is achieved with the weighted microbenchmarks compared to the non-weighted ones. On the application front, the total improvement depends on both the size of the underlying communicated messages, as well as the share of GPU communications in the total application runtime. More improvements are achieved with larger messages and higher share of GPU communications.

For future work, we intend to evaluate our proposed scheme with other GPU applications. We are specifically interested to see the results for applications that induce heavy GPU communications and/or impose irregular communication patterns among the GPUs. Utilizing other mapping algorithms to find potentially better GPU assignments is another venue for future work. We also seek to develop a hybrid metric that can precisely capture various characteristics of the GPU communication channels within a node.

Acknowledgment

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant #RGPIN/05389-2016, Canada Foundation for Innovation and Ontario Innovation Trust Grant #7154. Computations were performed on the Helios supercomputer installed at Universitè Laval and managed by Calcul Qubec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), NanoQubec, RMGA and the Fonds de recherche du Qubec - Nature et technologies (FRQ-NT). We especially thank Maxime Boissonneault for his technical support.

References

[1] The TOP500 June 2016 List, (https://www.top500.org/list/2016/06/). [Online; last accessed 10/14/2016].
[2] C. Obrecht, F. Kuznik, B. Tourancheau, J.-J. Roux, Scalable lattice boltzmann solvers for CUDA GPU clusters, Parallel Comput. 39 (6) (2013) 259–270.
[3] J. Glaser, T.D. Nguyen, J.A. Anderson, P. Lui, F. Spiga, J.A. Millan, D.C. Morse, S.C. Glotzer, Strong scaling of general-purpose molecular dynamics simulations on GPUs, Comput. Phys. Commun. 192 (2015) 97–107.
[4] H.C. Edwards, C.R. Trott, D. Sunderland, Kokkos: enabling manycore performance portability through polymorphic memory access patterns, J. Parallel Distrib. Comput. 74 (12) (2014) 3202–3216.
[5] M.J. Abraham, T. Murtola, R. Schulz, S. Páll, J.C. Smith, B. Hess, E. Lindahl, GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers, SoftwareX 1 (2015) 19–25.
[6] T. Gysi, C. Osuna, O. Fuhrer, M. Bianco, T.C. Schulthess, STELLA: A domain-specific tool for structured grid methods in weather and climate models, in: Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC '15, 2015, pp. 41:1–41:12.
[7] MPI3.1, (http://www.mpi-forum.org/docs/mpi-3.1/). [Online; last accessed 10/14/2016].
[8] I. Faraji, A. Afsahi, GPU-aware intranode MPI allreduce, in: Proc. 21st European MPI Users' Group Meeting (EuroMPI/ASIA'14), 2014, pp. 45–50.
[9] F. Ji, A.M. Aji, J. Dinan, D. Buntinas, P. Balaji, R. Thakur, W.-c. Feng, X. Ma, DMA-assisted, intranode communication in GPU accelerated systems, in: Proc. 14th International Conference on High Performance Computing and Communication & 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012, pp. 461–468.

[10] S. Potluri, H. Wang, D. Bureddy, A.K. Singh, C. Rosales, D.K. Panda, Optimizing MPI communication on multi-GPU systems using CUDA inter-process communication, in: Proc. Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2012, pp. 1848–1857.
[11] Open MPI, (http://www.open-mpi.org/). [Online; last accessed 10/14/2016].
[12] J.A. Anderson, C.D. Lorenz, A. Travesset, General purpose molecular dynamics simulations fully implemented on graphics processing units, J. Comput. Phys. 227 (10) (2008) 5342–5359.
[13] I. Faraji, S.H. Mirsadeghi, A. Afsahi, Topology-aware GPU selection on multi-GPU nodes, in: Proc. International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016.
[14] W. Gropp, K. Buschelman, FPMPI-2 fast profiling library for MPI. 2016 [Online; last accessed 10/14/2016].
[15] MVAPICH2, (http://mvapich.cse.ohio-state.edu). [Online; last accessed 10/14/2016].
[16] F. Pellegrini, J. Roman, Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs, in: Proc. International Conference on High-Performance Computing and Networking, 1996, pp. 493–498.
[17] G. Karypis, V. Kumar, A fast and high quality multilevel scheme for partitioning irregular graphs, SIAM J. Sci. Comput. 20 (1) (1998) 359–392.
[18] G. Karypis, V. Kumar, A parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distrib. Comput. 48 (1) (1998) 71–95.
[19] B. Goglin, Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc), in: Proc. International Conference on High Performance Computing & Simulation (HPCS), 2014, pp. 74–81.
[20] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, R. Namyst, HWLOC: A generic framework for managing hardware affinities in HPC applications, in: Proc. 18th Euromicro Conference on Parallel, Distributed and Network-based Processing, 2010, pp. 180–186.
[21] Infiniband Trade Association (IBTA), (http://www.infinibandta.org/). [Online; last accessed 10/14/2016].
[22] NVIDIA management library, (https://developer.nvidia.com/nvidia-management-library-nvml). [Online; last accessed 10/14/2016].
[23] M. Martinasso, G. Kwasniewski, S.R. Alam, T.C. Schulthess, T. Hoefler, A PCIe Congestion-aware Performance Model for Densely Populated Accelerator Servers, in: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, in: SC'16, IEEE Press, 2016, pp. 63:1–63:11.
[24] T. Lutz, C. Fensch, M. Cole, PARTANS: An autotuning framework for stencil computation on multi-gpu systems, ACM Trans. Archit. Code Optim. 9 (4) (2013) 59:1–59:24.
[25] A. Bhatele, L.V. Kalé, An evaluative study on the effect of contention on message latencies in large supercomputers, in: Proc. International Symposium on Parallel & Distributed Processing (IPDPS), 2009, pp. 1–8.
[26] P. Balaji, R. Gupta, A. Vishnu, P. Beckman, Mapping communication layouts to network hardware characteristics on massive-scale blue gene systems, Comput. Sci. 26 (3–4) (2011) 247–256.
[27] M.J. Rashti, J. Green, P. Balaji, A. Afsahi, W. Gropp, Multi-core and network aware MPI topology functions, in: Recent Advances in the Message Passing Interface, 2011, pp. 50–60.
[28] G. Mercier, E. Jeannot, Improving MPI applications performance on multicore clusters with rank reordering, in: Recent Advances in the Message Passing Interface, 2011, pp. 39–49.
[29] S.H. Mirsadeghi, A. Afsahi, PTRAM: A parallel topology-and routing-aware mapping framework for large-scale HPC systems, in: Proc. International Parallel and Distributed Processing Symposium Workshops (IPDPSW), 2016, pp. 386–396.