Topology-Aware GPU Selection on Multi-GPU Nodes

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

***Abstract*—GPU accelerators have successfully established themselves in modern HPC clusters due to their high performance and energy efficiency. To increase the GPU computational power in a cluster node and tackle larger problems, multi-GPU nodes have become the platform of choice for scientific applications. In a multi-GPU node, GPU devices are interconnected together via different communication channels. Thus, intranode inter-process communications among GPUs may traverse different paths with different latency and bandwidth capacity. As the number of GPUs within a multi-GPU node increases, the topology of GPU interconnects becomes more hierarchical, effectively increasing the heterogeneity of the GPU communication channels.**

In this paper, we provide evidence that the performance of different intranode GPU communication channels can be considerably different from each other. This is specially true for larger message sizes. Taking this into account, our goal in this work is to efficiently assign the available GPU devices on a multi-GPU node to MPI processes in order to improve the GPU-to-GPU communication performance. We tackle this challenge by proposing a topology-aware GPU selection scheme. Our scheme is capable of efficiently mapping MPI processes to the available intranode GPU devices, in a way that more intensive inter-process GPU communications take place on the more efficient communication channels. Our experimental results show that our topology-aware GPU selection scheme can improve the communication performance of the microbenchmarks with different communication patterns, specifically those with weighted and asymmetrical communications.

***Keywords*-Multi-GPU node; MPI; Topology-aware GPU Selection; Mapping**

I. Introduction

GPUs accelerators have established themselves in modern HPC clusters. The high compute capacity and the lower power dissipation provided by these accelerators is what the HPC systems are striving for. In fact, GPUs have been deployed in many supercomputers in the Top500 [1]. Moreover, multi-GPU nodes are increasingly gaining popularity among the state-of-the-art HPC clusters. Such nodes can provide higher GPU computational power and memory to the HPC scientific applications. The architecture of these nodes usually consists of multi-core processors and multiple GPU devices linked via different communication channels.

The Message Passing Interface (MPI) [2] is the de-facto standard for parallel programming in clusters. 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 reside on the GPU memory. To perform such communications efficiently, MPI must be tuned and become GPU-aware. In MPI applications, both the intranode and internode inter-process GPU communications are considered to be a major performance bottleneck. As a matter of fact, the overhead of the inter-process GPU communications in applications with high data interdependency may even wipe out the potential benefits of GPU offloading. In this regard, researchers have started to address the GPU communication bottleneck, by incorporating GPU-awareness for MPI point-to-point and collective operations [3], [4], [5].

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-to-GPU communications may have to traverse different paths. The intranode traversal path may consists of different communication channels, such as a single or multiple PCIe internal switches, a PCIe host bridge, or an inter-socket interconnect such as QPI. In this regard, we first present a series of tests on the multi-GPU node and show that the performance of these communication paths varies distinctively, specially as the message sizes increase. Taking this into account, our goal is to efficiently assign GPU devices to MPI processes in order to improve their GPU communication performance.

In this paper, we propose a topology-aware GPU selection scheme in order to assign GPU devices to MPI processes based on the GPU-to-GPU communication pattern and the physical characteristics of a multi-GPU node. We use three different metrics: latency, bandwidth, and communication distance to reflect the efficiency of different GPU-to-GPU traversal paths in a multi-GPU node. We also profile the MPI program to extract the corresponding GPU communication pattern. With the extracted information, 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 with more efficient connections. We use the SCOTCH [6] library to build the virtual and physical topology graphs and perform the mapping.

We have implemented our design into the Open MPI library [7]. For performance evaluations, we use four microbenchmarks: *2D Stencil*, *2D Torus*, *3D Torus*, and *4D Hypercube*. In all these microbenchmarks, the processes are first organized into a logical n-dimensional grid and communicate with their neighbors along each dimension of the grid. Moreover, we consider both symmetric and asymmetric communication patterns. In the symmetric case, all processes communicate with the same number of neighbors, whereas the number of neighbors for different processes can vary in the asymmetric case. We also consider both the weighted and non-weighted instances of each microbenchmark. While in the non-weighted case communication volume along all dimensions is identical, in the weighted case, we consider a relatively higher communication volume along one of the dimensions (i.e., the weighted dimension). According to the experimental results, our topology-aware GPU selection scheme can highly benefit the weighted instances of microbenchmarks (by up to 59%). We can also observe (to a lower extent) improvements for the non-weighted but asymmetric instances of microbenchmarks.

In this paper we make the following contributions:

- We use three metrics (latency, bandwidth, and communication distance) to reflect the performance of different intranode GPU communication channels in a multi-GPU node. The results show a considerable variation in communication performance among different GPUs, specially for larger messages.
- We propose a topology-aware GPU to process assignment policy that would take into account the GPU communication pattern and the physical topology of the multi-GPU node to improve the communication performance among the GPUs. We have integrated the proposed GPU scheme into the Open MPI library. To the best of our knowledge, this is the first application of topology awareness for GPU to MPI process assignment on multi-GPU nodes.
- We evaluate the performance of our proposed scheme with various microbenchmarks that model different communication patterns, and show that our scheme can highly benefit microbenchmarks with asymmetric and weighted communication patterns.

The rest of this paper is organized as follows. In Section II, we provide the background material. We present the motivation behind this work in Section III. We propose our topology-aware GPU selection scheme in Section IV. In Section V, we present and analyze our experimental results. We present the related work in Section VI. Finally, in Section VII, we conclude the paper and outline some future directions.

II. Background

In this section, we briefly discuss MPI [2], as well as the SCOTCH [6] and the NVML [8] libraries that we use in this paper for graph mapping and extracting the intranode GPU topology levels, respectively.

A. 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 well-known MPI implementations such as MVAPICH2 [9] and Open MPI [7]. 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.

B. 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. In particular, the SCOTCH library provides various APIs for graph partitioning and graph mapping. SCOTCH is capable of mapping any source graph onto any target graph. These graphs may have any topology, and their vertices and edges can also be weighted. In this work, we use the SCOTCH library to perform the mapping stage in our topology-aware GPU selection scheme.

C. NVIDIA Management Library

Information about the the internal architecture of a node can be extracted by various tools. In particular, HWLOC [10] 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 [11] or GPUs. In this work, however, we only use the NVIDIA Management Library (NVML) [8] to extract the GPU topology level of a multi-GPU node. The NVML library includes a set of C-based APIs that can be used for extracting various states and characteristics of the NVIDIA GPU devices, including monitoring, managing, and querying the GPU states and topology information. We use the $nvmlDeviceGetTopologyCommonAncestor()$ API function in the NVML library to retrieve the common ancestor in a GPU pair. The retrieved common ancestor is a value of type $nvmlGpuTopologyLevel\_t$ which represents the node level relationship between two GPUs. In general, the larger this value, the higher the topology level between the GPUs.

III. Motivation

In this section, we first discuss the various GPU topology levels with different communications channels in a multi-GPU node, and then evaluate their corresponding latency and bandwidth performance.

A. Intranode GPU Level

In a multi-GPU node, various traversal paths may exist among different GPUs. Therefore, communication between different GPUs 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 V). According to the figure, GPUs can communicate with each other through different paths. In particular, there are four different topology levels which we list them from **Level 0** to **Level 3**. Different topology levels include different communication channels, such as only a single PCIe internal switch in **Level 0**, or multiple PCIe switches and an inter-socket interconnect in **Level 3** (e.g., QPI). Such a variety of communication channels in different topology levels (as shown in Fig. 1) can lead to different GPU communication latency and bandwidth performance.

B. Impact of GPU Level on Intranode GPU-to-GPU Communication

To evaluate the latency and bandwidth capacity of various intranode GPU topology levels, we perform intranode ping-pong and bi-directional bandwidth tests. We examine both communication latency and bandwidth results 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. As can be seen in this figure, the GPU topology level has noticeable impact on the GPU communication latency and bandwidth for message sizes larger than 2KB. We can also observe that as the message size increases, the latency/bandwidth gap between different topology levels becomes wider. In general, we can infer that by increasing the GPU pair topology level, the latency increases and the bandwidth drops.

IV. 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.

We extract the GPU communication pattern by profiling the program in an initial run. In this regard, we instrument the MPI library to gather the GPU inter-process communications in a virtual topology matrix file. Using this file, we construct a GPU virtual topology graph. In this graph, vertices represent MPI processes, and edges represent the existence and the significance of GPU inter-process communications. Thus, the higher the edge value, the higher the communication volume between the associated communicating GPU peers.

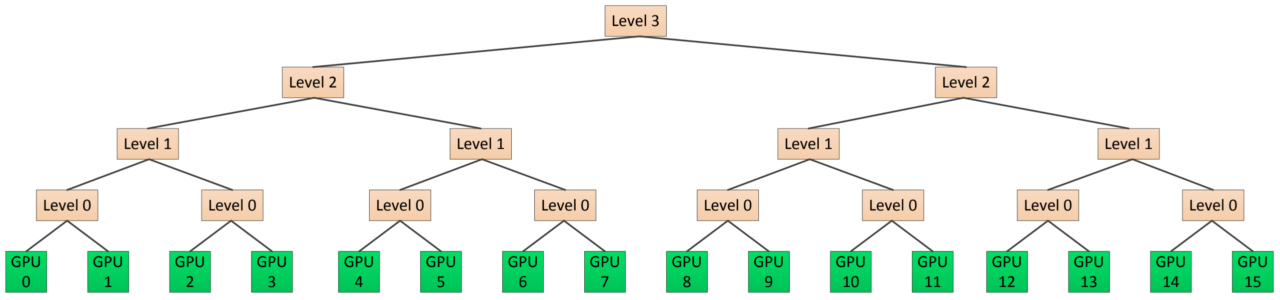
We leverage three different metrics to reflect the impact of different topology levels in a multi-GPU node: 1) latency; 2) bandwidth; and 3) communication distance. For each metric, we perform a series of tests to extract the associated physical topology matrix file (note that none of the tests requires root access). Using the generated files, we construct the GPU physical topology graph. Vertices in this graph represent the GPU device indices, and edges represent the significance of the GPU pair connection. 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 program execution. The virtual topology matrix file requires to be generated once in an initial run. The physical topology matrix files are also required to be created once on each 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.a). For each GPU pair, we calculate the ratio of the latency in the topology **Level 3** to the latency in the associated topology level of the pair. We calculate this ratio for message sizes in which the communication latency is affected by the topology level (greater than 2KB). We store the average of the ratio values in the topology matrix file.

For the bandwidth-based metric, we use normalized GPU pair bandwidth values from the bi-directional bandwidth test results (Fig. 2.b). For each GPU pair, we first calculate the ratio of the bandwidth in the topology level of the pair to the bandwidth in topology **Level 3** for message sizes with varying bandwidth on different levels (greater than 2KB). We only consider this ratio for message sizes in which the communication bandwidth is affected by the topology level. 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 run a test and 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.



- **Level 0:** Path between GPU pairs traverses a PCIe internal switch
- **Level 2:** Path between GPU pairs traverses a PCIe host bridge
- **Level 1:** Path between GPU pairs traverses multiple internal switches
- **Level 3:** Path traverses a socket-level link (e.g., QPI)

Figure 1. Different intranode GPU pair levels

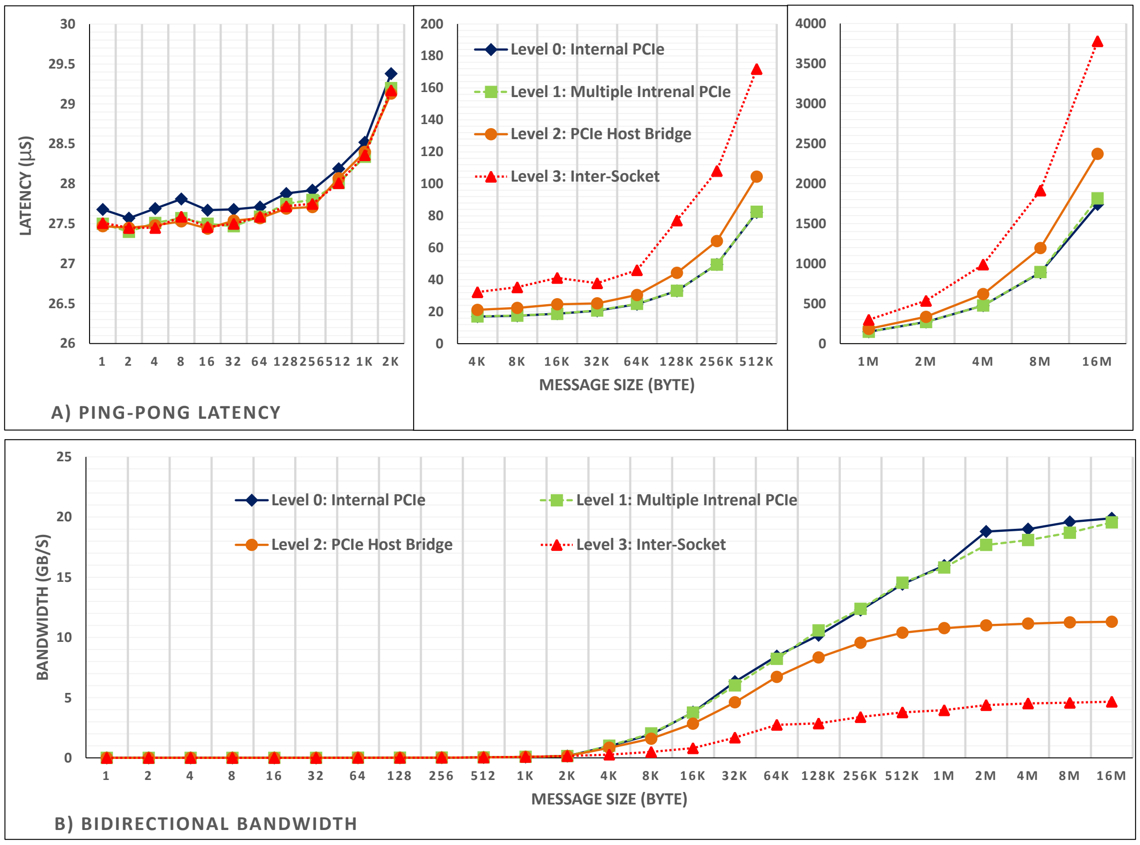


Figure 2. Effect of intranode GPU pair level on point-to-point latency and bi-directional bandwidth

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 (without loss of generality, process with rank 0) performs the mapping and scatters the mapping 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.

V. Performance Analysis

A. Experimental Setup

Our experiments are conducted on one node of the 6-node K80 Helios supercomputer installed in Université Laval. Each K80 node is equipped with 16 GPUs, 256 GB of memory, two Intel Xeon Ivy Bridge E5-2697 v2 with twelve 2.7 GHz cores (total of 24 cores per node) and runs a 64-bit CentOS 6.7. We conduct our experiments using Open MPI 1.8.8, CUDA 7.5, and SCOTCH 5.8 libraries. In all of our experiments, we consider 16 processes and a single process per GPU.

B. Experimental Results

In this section we compare the conventional GPU selection scheme against our proposed topology-aware schemes. In the conventional scheme, each process selects a GPU based on its associated rank number (usually process with rank i selects a GPU with index i). In our proposed scheme however, each process selects its GPU based on the topology-aware assignment results. Our scheme exploits three different metrics to extract the GPU physical topology graph, thus we consider three scenarios in our scheme: 1) latency-map; 2) bandwidth-map; and 3) distance-map.

To conduct our experiments we use four microbenchmarks that model different communication patterns: 1) *2D Stencil*; 2) *2D Torus*; 3) *3D Torus*; and 4) *4D Hypercube*. Two of the microbenchmarks have symmetric communication patterns (i.e., *2D Torus* and *4D Hypercube*), in the sense that all processes communicate with the same number of neighbors. We also consider both non-weighted and weighted cases in our microbenchmarks, with the weighted case having heavier communication along one dimension. Our first microbenchmark (*2D Stencil*) is a 5-point 2-dimensional Stencil, in which processes are first organized into a 2-dimensional mesh; each process then communicates with its immediate neighbors along each dimension. With 16 processes, the processes are logically organized into a 4×4 mesh (with no wraparound connections). In the weighted case, we carry out heavier communication along the x dimension. Our second microbenchmark (*2D Torus*) is a 2-dimensional torus, which is similar to the *2D Stencil*, but with wraparound connections. Our third microbenchmark (*3D Torus*), is a 3-dimensional torus, in which each process communicates with its neighbors along x, y, and z dimensions. With 16 processes, the processes are logically organized into a $4 \times 2 \times 2$ torus. In the weighted case, we carry out heavier communication along the longer dimension (dimension x). The fourth microbenchmark (*4D Hypercube*) is a 4-dimensional hypercube, in which each process communicates with its four neighbors along the x, y, z, and t dimensions. In the weighted case, we consider heavy communication on the dimension connecting the inner cube to the outer cube (dimension t).

In all of our microbenchmarks, we use the w parameter to determine the relative weight and consider w values of 1 and 3 for the non-weighted and weighted cases, respectively. A w value of 1 means the communications along all dimensions have the same weight, whereas in the case of $w = 3$, messages sent along one of the dimension will be 3 times larger than those sent along the other dimensions.

Our experimental results are the average of four runs and they are reported for the non-weighted and weighted microbenchmarks in Fig. 3 and Fig. 4, respectively. According to Fig. 3, for the non-weighted cases, we observe performance improvement in the *2D Stencil* and the *3D Torus* microbenchmarks, mainly for message sizes greater than 16KB. However, no (or limited) improvement is observed in the case of *2D Torus* and *4D Hypercube* microbenchmarks. We attribute this behavior to the fact that, with the conventional scheme, GPUs are already efficiently assigned to MPI processes, in a sense that most of the inter-process GPU communications will take place on the more efficient GPU traversal paths (i.e., lower topology levels). One reason behind this is due to the fact that both the *2D Torus* and *4D Hypercube* microbenchmarks, unlike the *2D Stencil* and *3D Torus*, have symmetric communication patterns, as all processes in these microbenchmarks are communicating with the same number of neighbor processes (4 neighbors in both *2D Torus* and *4D Hypercube*). On top of that, all communications in these microbenchmarks have the same weight. All in all, for the non-weighted symmetric microbenchmarks, there remains a limited potential for our topology-aware scheme to provide improvement.

On the contrary, in our weighted microbenchmarks, GPUs should be assigned to processes in a way that heavily communicating processes end up on GPU pairs with more efficient traversal path. According to Fig. 4, our GPU selection schemes can outperform the conventional GPU selection approach in most cases (with the only exception of 4KB to 32KB in the *3D Torus*). According to the figure, all of our schemes provide comparable performance results with each other. The only exception is for 16MB messages where the distance-based approach is mostly superior. It is also worth mentioning that in some cases the performance improvement of our schemes drop momentarily for some message sizes greater than 2KB. We attribute this to the eager/rendezvous protocol switching point (4KB) for the intranode GPU-to-GPU communication used in Open MPI. We could avoid the associated drop in performance improvement by increasing the default eager threshold; however, this may come at the expense of degrading the basic latency/bandwidth performance.

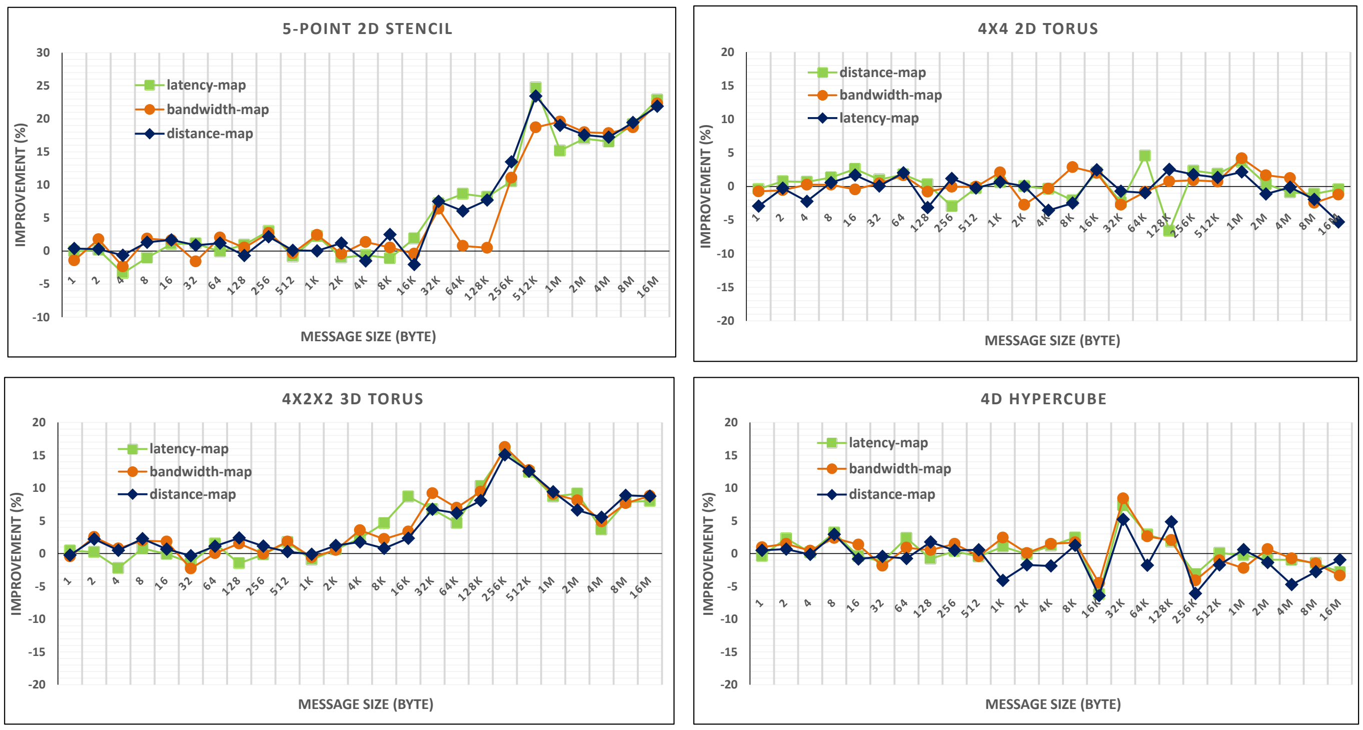


Figure 3. Runtime improvement of topology-aware mappings over default mapping on non-weighted microbenchmarks

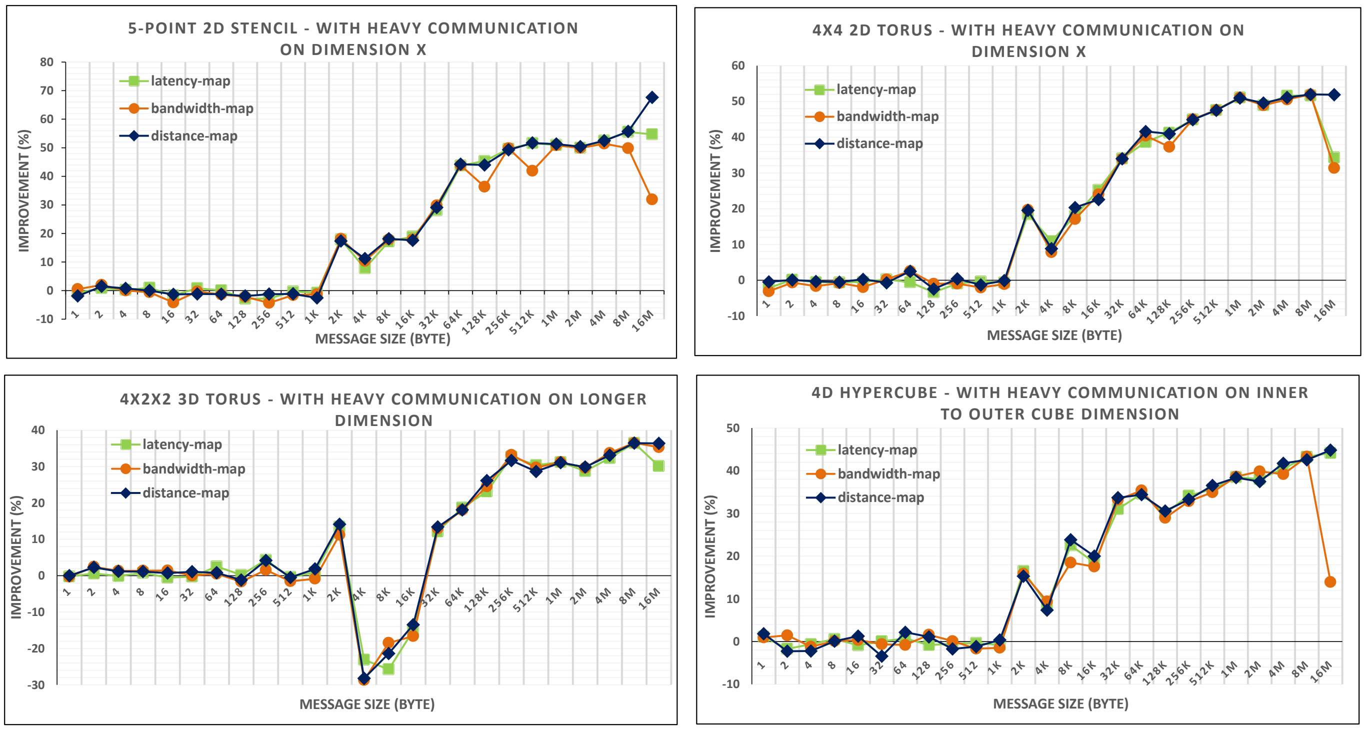


Figure 4. Runtime improvement of topology-aware mappings over default mapping on weighted microbenchmarks

In general, we observe that weighted microbenchmarks (with both symmetric and asymmetric communication patterns) benefit the most from our topology-aware GPU selection schemes. While our mapping schemes can also benefit the non-weighted asymmetric microbenchmarks, the performance of the non-weighted symmetric microbenchmarks mainly remain intact.

C. The Overheads

In this section, we analyze the overheads that are associated with our proposed topology-aware GPU assignment scheme. To this end, we measure the time it takes to do the mapping and compare it against the average time of a single communication iteration for each microbenchmark. Note that we do not consider the overhead of extracting the physical topology as it is required to be performed only once. We report the corresponding results in Fig. 5 for various message sizes. For each microbenchmark, Fig. 5 shows the minimum number of communication iterations that is required to compensate the overheads of topology-aware GPU assignment. As shown in the figure, the overheads become more and more negligible as the message size increases. For smaller messages, we might require up to 1000 communication iterations to get benefits from our topology-aware GPU assignment scheme. For messages larger than 2MB, however, the overheads are mitigated by only a single communication iteration. We can also observe that the 2D Torus and the 3D Torus microbenchmarks require the lowest and highest minimum number of communication iterations to compensate the overheads of topology-aware GPU assignment, respectively. This complies with the results shown in Fig. 4 where the improvements are higher for the 2D Torus microbenchmark than the 3D Torus.

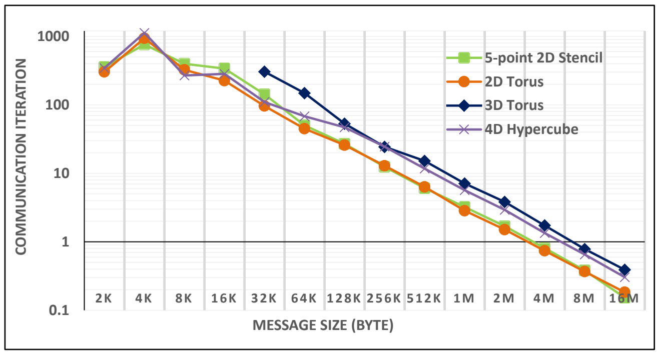


Figure 5. Number of communication iterations required to compensate the mapping overhead

VI. Related Work

In GPU clusters, researchers have studied various GPU-aware point-to-point and collective operations to improve the GPU communication performance [3], [4], [5]. In [3], the authors proposed an intranode GPU-aware MPI_Allreduce, in which CUDA IPC was used to gather the pertinent data into the shared GPU buffer, followed by an in-GPU reduction. The CUDA IPC copy type was also utilized in [4], [5] to improve one-sided and point-to-point communications.

Topology-aware mapping has also been extensively studied in the context of CPU-to-CPU communications. Several experiments carried out on large-scale systems such as IBM BG/P and Cray XT supercomputers have verified the adverse effects of contention and hop-count on message latencies [12]. Another study [13] shows that different mappings of an application on large-scale IBM BG/P systems can significantly affect the overall performance. Ma et al. [14] study the impacts of topology awareness and process distances on collective communications, and propose algorithms for constructing collectives whose communication patterns are optimized with respect to the physical topology of processes.

In [15], the authors propose a topology-aware mapping mechanism for two of the MPI topology functions, i.e., `MPI_Graph_create` and `MPI_Cart_create`. Virtual process topology of CPU communications is extracted through profiling the application, and is represented by an undirected graph structure with weighted vertices and edges. The normalized total communication between two processes is used as the metric for assigning edge weights. Since the `MPI_Graph_create` function does not support weighted edges, the authors use edge replication to represent edge weights. In addition, a weight is assigned to each vertex of process topology to represent the computation requirement of the corresponding process. For the physical topology, both node and network architectures are taken into account. HWLOC [16] is used to extract the architecture of multi-core nodes, whereas the *ibtracert* tool of InfiniBand subnet manager is utilized to discover network distances between all pair of nodes. The results are merged to build an undirected graph with both vertices and edges having weights. The weight of a given vertex represents the processing power of the corresponding processor whereas the weight of each edge represents the communication performance between the corresponding cores. The actual mapping is performed by the SCOTCH library.

Mércier and Jeannot [17] modify the implementation of the `MPI_Dist_graph_create` function in MPICH2 to provide it with a topology-aware reordering of processes. The HWLOC library is first used to gather hardware architecture information. One process on each multi-core node extracts the architecture of that node and sends it to a global root process. The whole hardware architecture of the system is represented in a tree structure in compliance with the hierarchical architecture of NUMA nodes. The authors implement an algorithm called TreeMatch to do the actual mapping. The algorithm extracts a tree from the matrix representing the communication graph and matches this tree to the hardware topology tree. It works recursively on each level of memory hierarchy (following a bottom-up approach) and groups processes in such a way that the cost of remaining communications is minimized.

In [18], another mapping approach is proposed which is similar to that of [15]. Using graph partitioning methods, the problem domain is decomposed so as to build a subdomain graph with a number of vertices equal to the number of cores in the system. The physical topology of the system is represented by a processor graph consisting of node graph and core graph. Core graph is built upon the information gathered by the HWLOC library with edge weights representing the logical distance between any pair of cores within a node. Node graph is a complete weighted graph that captures the internode topology of the system with the edge weights representing the bandwidth between any pair of nodes. Unlike the work done in [15] where the network distance between pair of nodes is used to assign edge weights, in this work the actual bandwidth between any pair of nodes is measured at the execution time. This is done by measuring the communication time of sending and receiving point-to-point data of different sizes between any pair of MPI processes residing on different nodes. Mapping is carried out in two steps. First, the sub-domain graph is partitioned into several groups and each group is mapped onto a vertex in the node graph. Next, the sub-domain in each group is mapped on the vertices in the core graph. SCOTCH is utilized to do the mapping (and partitioning) in both steps.

Rodrigues et al. [19] also use bandwidth measurement to determine the edge weights of the physical topology graph. In particular, a ping-pong microbenchmark is used to measure the bandwidth between each pair of cores in the system. The communication speeds generated by the ping-pong microbenchmark are then used to assign the edge weights in the physical topology graph. Application communication pattern is also represented by a weighted graph which is extracted by profiling a previous run of the application on the target system. The mapping step exploits the Dual Recursive Bipartitioning method (a graph partitioning method) provided by the SCOTCH library. The algorithm recursively partitions both the process and physical topology graphs into two disjoint sets such that the communication weight between the two sets is minimized in the process graph, and strongly connected processors are kept in the same set in the architecture graph. In each step, the resulting sets of processes are assigned to the resulting sets of processors. This way, heavily communicating processes end up on strongly connected processors.

As discussed above, mapping and topology awareness have been used for efficient CPU assignment in parallel systems. In this paper, we show that similar concepts apply to GPU communications, and topology awareness can be beneficial for GPU communications as well. Accordingly, we extract the communication pattern and the GPU topology of a multi-GPU node, and then use a mapping library to assign GPUs to MPI processes in order to improve the GPU communication performance.

VII. Conclusion And Future Work

In this paper, we showed that intranode GPU topology can affect communication performance. Accordingly, we proposed a non-trivial topology-aware GPU selection scheme that considers the program communication pattern and the physical characteristics 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 scheme can highly improve the communication performance of the weighted and asymmetric microbenchmarks (by up to 59%).

We are currently experimenting with some real applications to evaluate the performance impact of our topology-aware GPU selection schemes. As for the future work, we are interested in extending our work to across the cluster and evaluate the impact of the interconnection network level in our scheme.

Acknowledgment

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada Grant #RGPIN/238964-2011, Canada Foundation for Innovation and Ontario Innovation Trust Grant #7154. Computations were performed on the K80 Helios supercomputer installed in Université Laval and managed by Calcul Québec and Compute Canada. The operation of this supercomputer is funded by the Canada Foundation for Innovation (CFI), NanoQuébec, RMGA and the Fonds de recherche du Québec - Nature et technologies (FRQ-NT). We would like to especially thank Maxime Boissonneault for his technical support regarding our experiments.

References

[1] "The TOP500 November 2015 List," http://www.top500.org/lists/2015/11/.

[2] "MPI3.1," http://www.mpi-forum.org/docs/mpi-3.1/, [Online; last accessed 12/14/2015].

[3] I. Faraji and A. Afsahi, "GPU-Aware Intranode MPI Allreduce," in *EuroMPI/ASIA'14*, 2014, pp. 45–50.

[4] F. Ji, A. M. Aji, J. Dinan, D. Buntinas, P. Balaji, R. Thakur, W.-c. Feng, and X. Ma, "DMA-Assisted, Intranode Communication in GPU Accelerated Systems," in *HPCC'12*, 2012, pp. 461–468.

[5] S. Potluri, H. Wang, D. Bureddy, A. K. Singh, C. Rosales, and D. K. Panda, "Optimizing MPI Communication on Multi-GPU Systems Using CUDA Inter-Process Communication," in *IPDPSW'12*, 2012, pp. 1848–1857.

[6] I. Lütkebohle, "F. Pellegrini, Scotch and libScotch 5.1 User's Guide. Bacchus team, Universitt'e Bordeaux,," http://gforge.inria.fr/docman/view.php/248/7104/scotch_user\\5.1.pdf, 2008, [Online; last accessed 1/20/2016].

[7] "Open MPI," http://www.open-mpi.org/, [Online; last accessed 12/27/2015].

[8] "NVIDIA management library, 2015," https://developer.nvidia.com/nvidia-management-library-nvml, [Online; last accessed 12/27/2015].

[9] "MVAPICH2," http://mvapich.cse.ohio-state.edu, [Online; last accessed 12/27/2015].

[10] B. Goglin, "Managing the topology of heterogeneous cluster nodes with hardware locality (hwloc)," in *High Performance Computing & Simulation (HPCS), 2014 International Conference on*. IEEE, 2014, pp. 74–81.

[11] "Infiniband Trade Association (IBTA)," http://www.infinibandta.org/.

[12] A. Bhatele and 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.

[13] 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.

[14] T. Ma, T. Herault, G. Bosilca, and J. J. Dongarra, "Process distance-aware adaptive MPI collective communications," in *Proc. International Conference on Cluster Computing*, 2011, pp. 196–204.

[15] M. J. Rashti, J. Green, P. Balaji, A. Afsahi, and W. Gropp, "Multi-core and network aware mpi topology functions," in *Recent Advances in the Message Passing Interface*. Springer, 2011, pp. 50–60.

[16] 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. 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010, pp. 180–186.

[17] G. Mercier and E. Jeannot, "Improving mpi applications performance on multicore clusters with rank reordering," in *Recent Advances in the Message Passing Interface*, ser. Lecture Notes in Computer Science, 2011, vol. 6960, pp. 39–49.

[18] S. Ito, K. Goto, and K. Ono, "Automatically optimized core mapping to subdomains of domain decomposition method on multicore parallel environments," *Computers & Fluids*, May 2012. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0045793012001636

[19] E. R. Rodrigues, F. L. Madruga, P. O. a. Navaux, and J. Panetta, "Multi-core aware process mapping and its impact on communication overhead of parallel applications," in *Proc. Symposium on Computers and Communications*, 2009, pp. 811–817.