Hyper-Q Aware Intranode MPI Collectives on the GPU

Iman Faraji
Dept. of Electrical and Computer Engineering
Queen's University
Kingston, ON, Canada K7L 3N6
i.faraji@queensu.ca

Ahmad Afsahi
Dept. of Electrical and Computer Engineering
Queen's University
Kingston, ON, Canada K7L 3N6
ahmad.afsahi@queensu.ca

ABSTRACT

In GPU clusters, high GPU utilization and efficient communication play an important role in the performance of the MPI applications. To improve the GPU utilization, NVIDIA has introduced the Multi Process Service (MPS), eliminating the context-switching overhead among processes accessing the GPU and allowing multiple intranode processes to further overlap their CUDA tasks on the GPU and potentially share its resources through the Hyper-Q feature. Prior to MPS, Hyper-Q could only provide such resource sharing within a single process. In this paper, we evaluate the effect of the MPS service on the GPU communications with the focus on CUDA IPC and host-staged copies. We provide evidence that utilizing the MPS service is beneficial on multiple interprocess communications using these copy types. However, we show that efficient design decisions are required to further harness the potential of this service. To this aim, we propose a Static algorithm and Dynamic algorithm that can be applied to various intranode MPI collective operations, and as a test case we provide the results for the MPI_Allreduce operation. Both approaches, while following different algorithms, use a combination of the host-staged and CUDA IPC copies for the interprocess communications of their collective designs. By selecting the right number and type of the copies, our algorithms are capable of efficiently leveraging the MPS and Hyper-Q feature and provide improvement over MVAPICH2 and MVAPICH2-GDR for most of the medium and all of the large messages. Our results suggest that the Dynamic algorithm is comparable with the Static algorithm, while is independent of any tuning table and thus can be portable across platforms.

Keywords

GPU, MPI, Intranode collective, Hyper-Q, Interprocess copy

1. INTRODUCTION

The node architecture of modern High-Performance Computing (HPC) clusters consists of multicore processors and accelerators/co-processors. GPUs, among other accelerators, have shown to be a promising candidate in improving the performance per watt of the HPC clusters [9].

ESPM2015 November 15-20, 2015, Austin, TX, USA
Copyright 2015 ACM. ISBN 978-1-4503-3996-4/15/11 ...$15.00
DOI: http://dx.doi.org/10.1145/2832241.2832247.

Message Passing Interface (MPI) [3] is the de-facto standard for parallel programming in clusters. GPU communication and underutilization are considered as two of the major bottlenecks in GPU clusters. To address the communication bottleneck, researchers have started incorporating GPU-awareness for MPI point-to-point and collective operations [2, 7, 8]. However, to the best of our knowledge, no work has considered the effect of improving GPU utilization on communication. In this paper, we consider the aforementioned bottlenecks and propose efficient design alternatives for intranode MPI collective operations with data in the GPU memory. Previously proposed GPU-aware collectives are either designed to stage the data into the host buffer, performing the collective operation, and potentially pipelining the communications [8], or use CUDA IPC [5] copy to gather the pertinent data into a GPU shared buffer for a follow-up reduction operation [2]. However, when multiple processes are sharing a single GPU, leveraging host-to-device/device-to-host or IPC copies would incur high context switching overhead which lowers the chance of different copy types to overlap with each other; in addition, the same copy types cannot share the copy bandwidth with each other. This is associated with the fact that NVIDIA GPUs use a Time-Sliced Scheduler which only allows a single process to take over the GPU resources and place its CUDA requests. Therefore, communications from different processes to the GPU compute and memory engines on a single GPU will be serialized. In this regard, NVIDIA has recently introduced a Multi Process Service (MPS) [6], which eliminates the context switching overhead and allows intranode processes to share intranode GPU resources concurrently. However, the effect of the MPS service on the GPU communication has not been considered and analyzed yet. Taking this into account, this paper makes the following key contributions:

- We present the effect of the MPS on intra-node point-to-point using the IPC and the host-staged copy types. We show, not only does the MPS benefit these copy types, it can potentially provide further improvement on a combination of them. We also show that the most efficient combination of these types varies across different message sizes.
- We propose a Static and a Dynamic algorithm for intranode MPI_Allreduce. Both designs are tuned to exploit the MPS and use a combination of different copy types to perform their collective. The Static algorithm decides the number and type of the copy for interprocess communica-

tions based on a tuning table that is provided prior to the runtime. While the Dynamic algorithm chooses the number and type of the copy based on the runtime information.

Our proposed algorithms can be applied to all collectives. They are designed to efficiently leverage the MPS and Hyper-Q features. The experimental results on MPI_Allreduce show that with the MPS service enabled, our designs outperform MVAPICH2 and the MVAPICH2-GDR for most of the medium and all large messages. The rest of the paper is organized as follows. We provide the background material in Section 2. In Section 3, we present the motivation for this work. In Section 4, we propose our designs. We present our results in Section 5 and discuss the related work in Section 6. Finally, in Section 7, we make concluding remarks.

2. BACKGROUND

Intranode Interprocess GPU Communication: The host-staged and the CUDA IPC are two data copy types that can be used for GPU interprocess communications within a node. The host-staged copy requires to stage the data into an intermediate host buffer. The NVIDIA CUDA IPC copy [5], on the other hand, can be directly performed between the GPU address spaces of different intranode processes. The CUDA IPC copy, however, requires a process to expose a portion of its address space to other intranode process(es). Having an asynchronous nature, the IPC copies would require synchronization between their involved processes to guarantee their completion.

GPU-Aware MPI_Allreduce: MPI collective operations can be implemented in a hierarchical or non-hierarchical fashion, with the hierarchical designs going through intranode and internode phases. With the availability of large multicore nodes and the ever increasing use of GPUs, the intranode collectives play a crucial role in the overall performance of the cluster-wide collective operations. MPI_Allreduce is an example of such collective operations that is extensively used in many MPI applications. MVAPICH2 and MVAPICH2-GDR [4] are two popular implementations of the MPI standard. MVAPICH2 follows a general design for most GPU collectives (including MPI_Allreduce), in which all processes first stage their data into the host buffers, then the collective operation is performed on them, and the result is eventually stored into the GPU memory. MVAPICH2-GDR is a proprietary design of the MPI standard for GPU clusters. MVAPICH2-GDR utilizes GPUDirect RDMA for internode and loopback/gdrcopy for intranode communications to boost the performance of the GPU point-to-point and collective operations, specially for short messages.

3. MOTIVATION

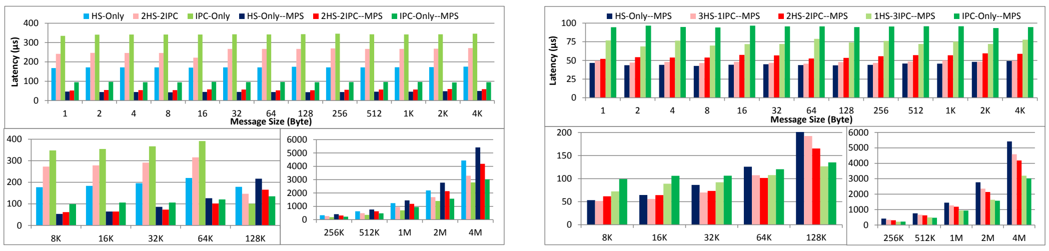
Multi Process Service and Hyper-Q: Hyper-Q is an NVIDIA feature that provides potential concurrency among CUDA tasks from a single process. However, Hyper-Q by itself cannot provide concurrency among CUDA requests from multiple processes to the GPU compute and memory engine. In order to provide such concurrency across multiple processes, NVIDIA has introduced the Multi Process Service (MPS) [6] for GPUs with compute capability of 3.5 and above. The MPS service acts as a funnel to collect CUDA tasks from multiple intranode processes and issue them to the GPU as if they are coming from a single process so that the Hyper-Q feature can take effect. With MPS enabled, there is only a single context (i.e., MPS context) on the GPU. This allows all processes to share the GPU resources and eliminate the context switching overhead.

Impact of MPS and Hyper-Q on Communication: While the effect of Hyper-Q through the MPS service is evaluated on offloaded computational kernels [10], in this section, for the first time, we evaluate the impact of the Hyper-Q feature and the MPS service on point-to-point intranode communication. We consider four point-to-point communicating pairs that are first synchronized and then perform pair-wise communications with either the host-staged (HS) or CUDA IPC data copy method. We consider three scenarios: 1) all communications are performed with only HS copy; 2) half of the communications are performed using HS and the other half are performed using CUDA IPC; and 3) all communications are performed using CUDA IPC. Fig. 1.a presents the microbenchmark results with and without the MPS service on our experimental platform described in Section 5. Three main observations can be made. The most apparent observation is that both host-staged and IPC copy types can benefit from the MPS for small and medium messages. For 128KB messages and above, however, the MPS service imposes overhead on both copy types, with a larger impact on the host-staged copy. Secondly, it can be seen that the host-staged copies (with or without MPS) are faster than the CUDA IPC copies for small and medium size messages. The opposite trend is seen for large message sizes, in which the CUDA IPC copies (with or without MPS) are superior to the host-staged copies. Finally, It can be observed that in some message sizes (32KB and 64KB) with MPS enabled, the communication using both copy types is superior to the communication with a single copy type. This implies that the Hyper-Q feature, through the MPS service, is providing some overlap between the host-staged and CUDA IPC communications. This phenomenon is shown to exist for a larger message range (8KB to 256KB) when all possible combination of the copy types are evaluated under MPS (Fig. 1.b). Taking these results into account, it can be argued that there is no silver bullet combination of different copy types working efficiently across all message sizes.

4. DESIGN AND IMPLEMENTATION

Static algorithm: The Static algorithm makes its decision based on a priori information from a tuning table. That associates the most efficient combination of the copy types to each message size and process count. The steps of the Static algorithm are listed below:

Step1. Gather: All processes copy their share of data into the GPU shared buffer area using a copy type that is assigned to them by the leader process (without loss of generality process with `rank 0` is considered as the leader process). It is noteworthy to mention that, having a single fixed leader is not a scalability concern in this design. In the case of the host-staged copy, there is only a single host-to-device and a single device-to-host engine on the GPU which cannot be shared among different processes (with or without MPS); in the case of the CUDA IPC, with only a single leader the copies can share the local GPU bandwidth and proceed concurrently on different streams (only with MPS); thus no room exists for further improvement by increasing the number of leaders. The leader is responsible for finding the most efficient combination of the copy types in the tuning table, assigning a particular copy type to each pro-



a. Selected Combinations With and Without MPS

b. All Combinations With MPS

Figure 1: Hyper-Q effect on intranode point-to-point communication

cess, and then querying their completion. **Step2. Kernel Function:** for MPI_Allreduce, an element-wise reduction is called on the aggregated data in the GPU shared buffer. For other collective types, this step may call another kernel function or be skipped. **Step3. Scatter:** The collective result is stored into the destination buffer of the participating processes. The leader process again assigns a particular copy type to each process based on the tuning table information.

Dynamic algorithm: Unlike the Static algorithm, the Dynamic algorithm is independent of any tuning parameters. It determines the type and number of copies for each process dynamically at runtime, based on their availability and efficiency by querying their responsiveness. Different steps of the Dynamic algorithm are listed below:
Step1. Gather: All processes copy their share of data into the GPU shared buffer of the leader process using an algorithm that goes through the following phases:

Phase1. Initialization: Considering that at this point no prior knowledge about the copy types exists, assessing the responsiveness of a copy type can only be done by actually assigning a host-staged and a CUDA IPC copy type to the first two (non-leader) processes arriving at the collective call and then querying their completion.

Phase2. Progress: The leader process queries the pending copies and waits until one of them completes. The intuition is that it is more efficient to issue multiple copies on a faster copy type all at once. In this regard, we calculate the difference between the number of completed communications between the two copy types. A zero or a negative difference indicates that the currently completed copy type is not as fast as the other copy type, thus only a single copy is issued with this slow but yet available copy type. A positive difference, on the other hand, indicates that the completed copy type is faster than the other type and thus multiple copies should be assigned to the next available processes. We determine the number of the issued copies to be two to the power of this difference; this way, we ensure quick assignment of the interprocess communications to the faster copy type and thus using it more frequently. This procedure continues until the last copy is issued.

Phase3. Final Copy Completion: For the final pending copy the leader takes a different approach. The rationale behind this is that, on one hand, the last pending copy could potentially linger for a long time; and on the other hand, there is no pending copy using the other type. At this point, the leader checks if there has been any successful completion of this copy type before. If this is not the case, the leader considers this copy type to be slow and assigns the final copy to be re-sent with the other copy type. If the slow copy turns out to be of the host-staged type, the remaining portion of the host-staged copy (if any) will be discarded. This can be supported by packetizing the host-staged copies into large chunks and sending them back to back; once a process receives a re-send assignment with the IPC copy type, the remaining packets of the host-staged copy will be discarded. However, this is not applicable for the IPC copies, thus we overlap them with the next steps of the collective.
Step2. Kernel Function: Similar to the Static algorithm.
Step3. Scatter: Both copy types can be potentially used to scatter the result. However, unlike `Step 1`, the leader now has some knowledge about the efficiency of the copy types. Thus, it goes through the following phases:

Phase1. Initialization: If there has been no successful completion of a copy type since the beginning of the collective operation, the leader tags it as a slow copy type and avoids using it in the scatter step. Otherwise, both copy types can potentially be used in this step.

Phase2. Progress: If all copies are initiated using a single type, the completion of that type is only required to be queried. Otherwise, the leader monitors the progress of both copy types and uses the faster copy type more frequently and waits for all of the pending copies in the current or previous step(s) to complete before returning form the collective.

5. PERFORMANCE EVALUATION

Our experiments are conducted on a 16-core node equipped with an NVIDIA K20M GPU. The node has a dual socket Intel XEON E5-2650 clocked at 2.0 GHz, a 64 GB of memory, and is running Fedora 19 and CUDA Toolkit 6.5. We present our results for MPI_Allreduce with and without the MPS and compare them against MVAPICH2-2.1 and MVAPICH2-GDR-2.0, using the OSU Micro Benchmark 4.3 [1].

In MPI_Allreduce, as shown in Fig. 2, the Hyper-Q feature through the MPS service improves all approaches except for MVAPICH2-GDR for small messages. With 16 processes and MPS enabled, the average speedup across all message sizes for the Static and Dynamic algorithms over the MPS disabled case is 3.23 and 2.77, respectively; the maximum speedup is 4.34 and 3.24, respectively. The overhead of using the Dynamic algorithm over the Static decreases as the message size increases and there are also a few cases in which the Dynamic algorithm is superior. We associate this to the fact that the static approach uses the integer values stored in the tuning table which are the rounded average of a thousand runs. While the Dynamic algorithm chooses the number and type of the copy in each run. Thus, it has the potential to be more accurate in choosing the right number and type of copy across multiple runs. The benefit of the Static algorithm over MVAPICH2 and MVAPICH2-GDR

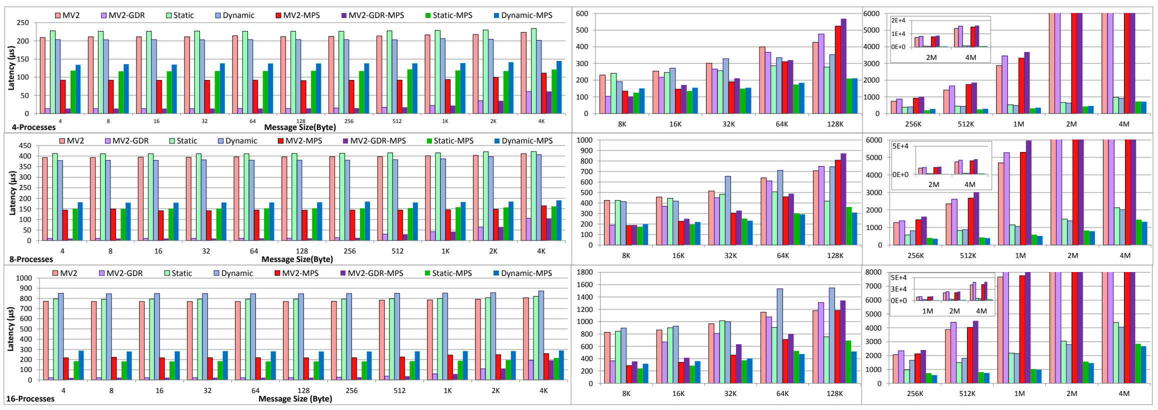


Figure 2: Hyper-Q effect on MVAPICH2, MVAPICH2-GDR, Static, and Dynamic approach w and w/o MPS

starts at 16KB, 8KB, and 8KB for 4, 8, and 16 processes, respectively. The Dynamic algorithm outperforms MVAPICH2 and MVAPICH2-GDR for messages greater than 16KB. With MPS, the average speedup over MVAPICH2-GDR on 16 processes for the Static and Dynamic algorithms is 2.66 (up to 14.28) and 2.79 (up to 15.23), respectively.

6. RELATED WORK

In GPU clusters, Singh et al. [8] leveraged host-staged copy types and optimized internode MPI_Alltoall by pipelining device-to-host and host-to-device CUDA memory copies with the network communications. In [2], we proposed intranode GPU-aware MPI_Allreduce, in which CUDA IPC is used to gather the pertinent data into the GPU shared buffer for a follow up in-GPU reduction. In this paper, however, we use a combination of both copy types for intranode interprocess communications and decide the number and type of the copies based on the information that is gathered either during or prior to the runtime. We also evaluate the effect of the Hyper-Q through MPS on our proposed designs.

Shi et. al. [7] proposed different techniques and leveraged gdr and loopback copy types to optimize the eager protocol in internode GPU-to-GPU communications for small messages. While their work targets small messages, our proposed algorithms also improve medium and large messages.

Wende et. al. [10] utilized the MPS to evaluate Hyper-Q on the GPU computational kernels from multiple processes; while our work evaluates the MPS and Hyper-Q feature on multiple intranode GPU interprocess communications with different copy types and proposes two algorithms that can efficiently use such features to maximize performance.

7. CONCLUSION

In this paper, we evaluated the impact of the recently introduced NVIDIA MPS service on the intranode host-staged and CUDA IPC interprocess copy operations. We observed that in multiple communications with these copy types, the Hyper-Q feature through MPS not only improves each of the copy types but also allows similar and different copy types to overlap with each other. Taking this into account, we proposed two design alternatives for intranode collectives: A Static Hyper-Q aware algorithm and a Dynamic Hyper-Q aware algorithm. These designs decide the number and type of the copies to be used in their interprocess communications based on different algorithms and types of information. While our designs can be applied to other MPI collective operations, as a test case we showed their effectiveness on MPI_Allreduce. The performance of the Dynamic approach is comparable with the Static approach, while it has the advantage of being independent of any tuning parameter and thus can be portable across different platforms. With the MPS enabled, both of our designs outperform MVAPICH2 and MVAPICH2-GDR for medium and large message sizes.

8. ACKNOWLEDGMENT

This work was supported by the Natural Sciences and Engineering Research Council of Canada Grant RGPIN/238964-2011, Canada Foundation for Innovation and Ontario Innovation Trust Grant 7154.

9. REFERENCES

[1] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda. OMB-GPU: A Micro-benchmark Suite for Evaluating MPI Libraries on GPU Clusters. In *EuroMPI'12*, pages 110–120, 2012.

[2] I. Faraji and A. Afsahi. GPU-Aware Intranode MPI Allreduce. In *EuroMPI/ASIA'14*, pages 45–50, 2014.

[3] MPI-3.1, http://www.mpi-forum.org/docs/mpi-3.1/ (Last accessed 8/14/2015).

[4] MVAPICH2, http://mvapich.cse.ohio-state.edu (Last accessed 7/27/2015).

[5] NVIDIA, "CUDA IPC", NVIDIA CUDA C Programming Guide Version 4.1, 2011.

[6] NVIDIA, "MPS", Sharing a GPU between MPI processes: Multi-Process Service - vR331, 2015.

[7] R. Shi, S. Potluri, K. Hamidouche, J. Perkins, L. Mingzhe, D. Rossetti, and D. Panda. Designing efficient small message transfer mechanism for inter-node MPI communication on InfiniBand GPU clusters. In *HiPC'14*, 2014.

[8] 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 *Cluster'11*, pages 420–427, 2011.

[9] The TOP500 June 2015 List, http://www.top500.org.

[10] F. Wende, F. Cordes, and T. Steinke. Multi-threaded Kernel Offloading to GPGPU using Hyper-Q on Kepler Architecture. In *Technical Report-ZIB*, 2014.