Large-Message Nonblocking MPI_Iallgather and MPI_Ibcast Offload via BlueField-2 DPU

Nick Sarkauskas, Mohammadreza Bayatpour
Tu Tran, Bharath Ramesh, Hari Subramoni, Dhabaleswar K. Panda
The Ohio State University
{sarkauskas.1,bayatpour.1,tran.839,ramesh.113,subramoni.1,panda.2}@osu.edu

***Abstract*—Since the introduction of nonblocking collectives in the MPI-3 standard, communication has been progressed by several mechanisms. One such mechanism includes modifying the application code to periodically call MPI_Test to enter the MPI library. Another launches an extra thread per core to progress communication asynchronously. Communication progression can also be offloaded to the Host Channel Adapter (HCA) using the latest hardware. In this paper, we explore this last option by using the Data Processing Unit (DPU) shipped with the BlueField-2 SmartNIC adapter to offload progression of nonblocking MPI_Ibcast and MPI_Iallgather collectives. For both collectives, we present several designs which take advantage of the DPU. We demonstrate the efficacy of our proposed designs through microbenchmark evaluations. At the microbenchmark level, total execution time of the osu_ibcast microbenchmark can be reduced by up to 54% using our DPU-based Ibcast designs. Total execution time of the osu_iallgather microbenchmark can be reduced by up to 43%. To the best of our knowledge, this is the first work to optimize nonblocking broadcast and allgather collectives on emerging BlueField DPUs.**

***Index Terms*—BlueField, SmartNIC, MPI, Collective, Offload**

I. Introduction and Motivation

Over the last decade, supercomputing systems have grown as a result of modern multi-core/many-core architectures and the availability of low-latency and high-bandwidth commodity interconnects such as InfiniBand [1] (IB). Running on such systems are scientific applications predominantly written with the Message Passing Interface (MPI) [2] parallel programming model. MPI offers point-to-point and collective primitives which allow the application programmer to orchestrate communication between processes in a convenient and effective manner.

Before the MPI-3 standard introduced nonblocking collective primitives, processes in a (blocking) collective needed to stay within the MPI library until their role in the communication was complete. However, programs modified to use nonblocking collectives will first initiate the communication, then perform useful computation which does not depend on the communication initiated, and then finally call the MPI_Wait() routine (or a variant of it) to finalize the communication. If there was enough overlap, the communication is completely hidden.

While the application is doing the useful computation, there must be progress on the communication, e.g. polling the underlying network fabric for completion notifications, responding to incoming control messages, managing internal MPI library buffers, etc. Progress can happen by the application itself calling MPI_Test() to reenter the MPI library, or it can be done by a separate entity such as an extra thread or a hardware mechanism.

Modifying the application to call MPI_Test() requires significant programmer effort to find the optimal location and frequency of calls within the application and is thus not desired. Using a separate thread to progress communication in the background is also not desired as it takes compute resources away from the application, requires context switching between threads, and is susceptible to OS noise.

Hardware mechanisms such as SHARP [3] and hardware tag-matching [4] are desirable since they reduce the amount of CPU resources needed to progress communication and thus increase amount of resources spent on useful scientific computation. However, SHARP can only offload reduction collectives, and hardware tag-matching cannot be used to progress collectives.

The NVIDIA/Mellanox BlueField-2 is the latest development in hardware offload mechanisms. The BlueField-2 is a programmable hardware offload solution for software-defined networking, virtualization, security, and storage within the datacenter. The Data Processing Unit (DPU) within the BlueField aims to be a new class of processor that is ideal for handling large amounts of incoming/outgoing data to the host, reclaiming CPU cycles and thus increasing the amount of compute available for end-users in the datacenter. Moreover, the DPU has been characterized in previous work [5] to transfer large amounts of data with the same communication latency as the host. Along with its offloading capabilities, this provides opportunities to explore designs which provide good communication performance as well as overlap of compute and communication.

II. Contributions

BlueField-2 is becoming widely adopted in new datacenter deployments because of its ability to offload managerial tasks such as access control, intrusion detection, online compression, and transport layer security (TLS) encryption using the on-chip dedicated ASIC. However, using the DPU to offload MPI communication is still nascent. In this paper, we take on this challenge by introducing novel nonblocking broadcast and allgather algorithms which take advantage of the DPU. To demonstrate the effectiveness of the algorithms, we compare

This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002

and provide a detailed analysis of results at the microbenchmark level using OSU MicroBenchmarks.

To summarize, this paper makes the following contributions:

- Identify bottlenecks and shortcomings of existing nonblocking MPI_Ibcast and MPI_Iallgather algorithms.
- Propose, design and implement novel nonblocking MPI_Ibcast and MPI_Iallgather algorithms using the DPU as an offload mechanism that provides maximal overlap while providing low pure communication latency comparable to CPU-based schemes.
- Analyze performance of the proposed DPU-based broadcast and allgather algorithms at the microbenchmark level with OSU MicroBenchmarks (OMB) [6].

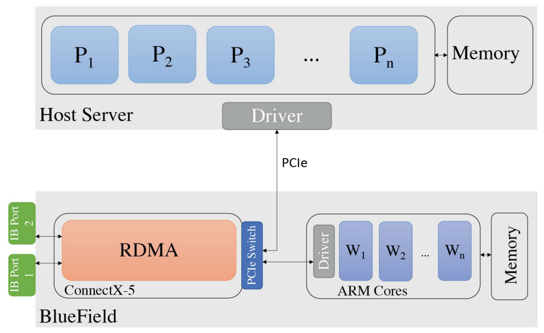


Fig. 1. BlueField Smart NIC Architecture (Courtesy [5])

III. Background

A. Capabilities of SmartNIC Adapters

SmartNIC adapters tend to come in three types: ASIC-based, FPGA-based, and multi-core microcontroller-based [7]. ASIC-based SmartNICs are cost-effective, but they are inflexible with regards to their function. FPGA-based SmartNICs are flexible, but are relatively inaccessible due to the prerequisite VHDL knowledge necessary to program them. Lastly, multi-core microcontroller-based SmartNICs such as the BlueField are shipped with a fully-programmable system-on-chip to accelerate arbitrary network functions. The cores are typically weaker (ARM or MIPS) in an effort to keep costs low.

B. BlueField-2 DPU

BlueField SmartNIC models come in two versions, BlueField SmartNIC for Ethernet and BlueField SmartNIC for Ethernet and Infiniband. The latter model uses Mellanox's Virtual Protocol Interconnect (VPI) to support incoming packets of either Infiniband or Ethernet protocols. The latest generation (BlueField-2) ships with the ConnectX-6 network controller. Versions of the DPU can be further categorized into Crypto-Enabled and Crypto-Disabled. A Crypto-Enabled DPU will include an ASIC for accelerating public-key cryptography as well as a random number generator.

Figure 1 shows a block diagram of the BlueField architecture. The DPU is a system-on-chip with 8 mesh-connected 64-bit ARMv8 A72 cores, 16 GB of DDR4 memory, a ConnectX network controller, and an integrated PCIe switch.

Each DPU can be configured to run in one of two modes of operation: Separated Host and Embedded CPU Function Ownership (SmartNIC) mode. Separated Host mode configures the ARM cores to behave as if they were another host on the network entirely. The ARM cores and the x86 host both share the same PCIe physical function and thus share bandwidth. In this paper, we base our designs off of the BlueField DPU with both ports configured to Separated Host mode.

Embedded CPU Function Ownership mode is an alternative configuration that forces packets going to the host to be first routed through the ARM cores for processing. The physical PCIe function is controlled by the ARM cores and can be used to offload common datacenter tasks like access control and packet inspection. [8].

C. BluesMPI Communication Offload Framework

Processes running on the BlueField-2 DPU set up in Separated Host mode are roughly analogous to processes running on the host, only they are running on the SmartNIC ARM cores (DPU) rather than the host CPU. Since all processes are shielded from unauthorized access from one another on the HCA via InfiniBand Protection Domains, it is not possible to directly issue RDMA commands on the DPU which operate on memory that was registered by processes on the host. This is because during normal operation, an InfiniBand HCA will return an lkey for local access and an rkey for remote access when the host registers a memory region with it. In order for the DPU to directly progress communication on behalf of the host, it would need to post verbs operations using the lkey of the host process' memory region. Since this lkey was intended for the Protection Domain associated with the host process, this would return an access violation error. Therefore, in our previous work in BluesMPI offload framework [5], we send task objects containing the buffer address and rkey information of all buffers (send and receive on all processes) on each host to each process running on all DPUs. The DPU processes are programmed to unpack the task object, read the buffer address and rkey information, and finally perform the required communication on behalf of the host by first reading the data into a temporary staging buffer in DPU memory and then writing it to the destination. The DPU spends a vast majority of its runtime in this last step performing the RDMA read and write operations to orchestrate communication. This fosters a large design space that is the subject of this paper.

IV. Proposed MPI_Ibcast Designs

In this section, we detail our proposed nonblocking DPU-based broadcast algorithms.

A. Proposed Flat MPI_Ibcast

In the Flat algorithm for MPI_Ibcast, we program the DPU to carry out the following steps once it receives the buffer addresses and key information it needs to progress communication. Since the exchange of buffer information has an overhead, we target large message sizes where collectives are bandwidth-bound. Figure 2 illustrates the steps in this algorithm.

1) Read Stage: DPU rank 0 (in DPU world) will perform an RDMA read on the host root's sendbuf, staging the buffer in its memory.

2) Blocking Broadcast Stage: Blocking MPI_Bcast is then called on DPU world (a communicator with all DPU processes in the job) to stage the buffer on all DPUs. Since this broadcast is only among the DPUs, the hosts are still doing application computation.

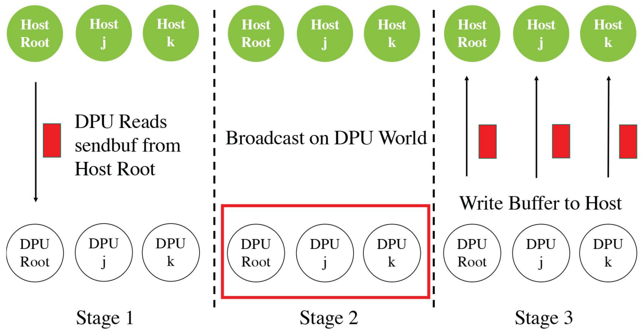


Fig. 2. Proposed Flat Broadcast With 3 Host Processes

3) Writeback to host: When the blocking MPI_Bcast finishes, the buffer is ready on each DPU and now must be written back to the host processes recvbuf. The DPUs will then perform these RDMA writes to the processes on its host.

4) Completion Notification: The DPU processes then perform a final RDMA write to set a completion flag on each of its host processes. The hosts poll on this flag during MPI_Wait until it is set. Once it is, they exit the MPI library and resume executing the application.

This method is appealing because each of the blocking broadcast algorithms within MVAPICH2-X has been well-researched and highly optimized since the inception of the project.

B. Proposed Hierarchical MPI_Ibcast

In the hierarchical algorithm for MPI_Ibcast, we program the DPU to perform these steps which are similar to the flat algorithm except that the DPUs write only to node-level leaders, and the host processes perform a memcpy in MPI_Wait:

1) Read Stage: Like the flat algorithm, DPU rank 0 will perform an RDMA read on the host root's sendbuf into DPU memory.

2) Blocking Broadcast Stage: Blocking MPI_Bcast is then called on DPU world. In MVAPICH2-X, scatter allgather blocking broadcasts are chosen for large messages. Binomial tree based broadcast is used for small messages.

3) Writeback to Node-Leaders: The buffer staged on each DPU is now RDMA written to a shared memory region on node-level leaders.

4) Completion Notification: The DPU processes then perform an RDMA write to set the completion flag on each of its host processes.

5) Copy from Shared Memory to Recvbuf: Once the completion flag is set, the hosts can start copying the buffer from shared memory to their respective recvbufs. This step exclusively takes place during MPI_Wait. After the memcpy completes, the broadcast is finished.

V. Proposed MPI_Iallgather Designs

In this section, we detail our proposed DPU-based MPI_Iallgather algorithm. We propose two designs (flat and hierarchical) to implement the collective.

A. Proposed Flat MPI_Iallgather

For the flat allgather, we present a similar algorithm to the flat broadcast. The allgather version is modified to read each host process' unique contribution to the final allgather recvbuf.

In this design, each sendbuf is buffered in the DPUs with an initial RDMA read. The DPUs will then write directly to the receive buffer of every host process in a round-robin fashion starting from its own host to avoid network contention. The algorithm is thus broken down into the following stages:

1) Initial Read: Each DPU process begins by reading the sendbuf at each one of the host processes running on its node, for a total of *processes per node (PPN)* RDMA reads. This is akin to a node-level gather to the DPU.

2) Write to Recvbuf: For each host process, the DPU writes the gathered buffer into the appropriate index into the recvbuf. For example, in a 2 PPN job, the DPU responsible for host ranks 0 and 1 would write the combined buffers to indices 0 and 1 in each host process' recvbuf. The DPU responsible for host ranks 2 and 3 would write directly to indices 2 and 3, and so on. Since each DPU does this same task for different ranks, the recvbuf has the complete allgathered buffer once they finish. Figure 3 illustrates this step with 3 host processes. In the figure, each color represents the sendbuf for that node in a 1 PPN job, or the gathered buffer for that node in a multiple PPN job.

3) Completion Notification: The DPUs will send a completion notification to each of their host process using another RDMA write. Once the hosts see the flag is set during the call to MPI_Wait, they are free to return to the application.

B. Proposed Hierarchical MPI_Iallgather

The following steps illustrate the proposed hierarchical DPU-based allgather algorithm.

1) Read Stage: Each DPU process reads the send buffer at each one of the host processes running on the node, for a total of *PPN* RDMA reads. This is akin to a node-level gather to the DPU.

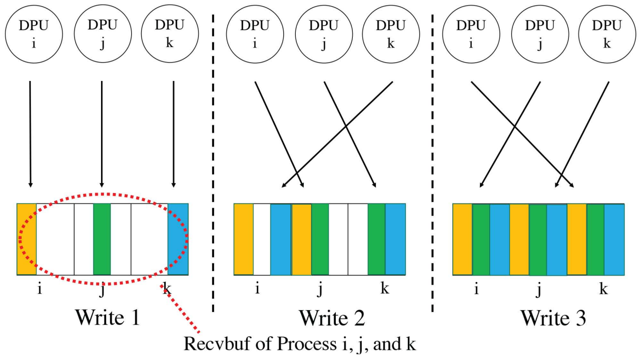


Fig. 3. Write to Recvbuf Step of the Proposed Flat Allgather Algorithm with 3 Processes

2) Writeback to Node-Leaders: For each node-level leader, the DPU writes the gathered buffer into the appropriate index into that leader's shared memory. In a 2 PPN job for example, the DPU responsible for host ranks 0 and 1 would write the combined buffers to indices 0 and 1, only instead of directly to the recvbuf (like in the flat algorithm), it goes in each node-level leader's shared memory. Since each DPU does this same task for different ranks, shared memory has the complete allgathered buffer once all of them finish.

3) Completion Notification: The DPUs will send a completion notification to each of their host process using another RDMA write.

4) Copy from Shared Memory to Recvbuf: During MPI_Wait, each host process will then copy the allgathered buffer in parallel to its own receive buffer, completing the collective.

VI. MicroBenchmark Level Evaluations

In this section, we discuss the experimental results of running the MPI_Iallgather and MPI_Ibcast primitives using OSU MicroBenchmarks (OMB) [6] for 32 node jobs using up to 32 processes per node (PPN), for a maximum of 1024 process jobs.

The proposed algorithms discussed in IV and V are implemented inside the MVAPICH2-X v2.3 MPI library with the BluesMPI framework [5] for sending buffer information (addresses and rkeys) to each DPU. We compare our designs with the default nonblocking allgather and broadcast algorithms from MVAPICH2-X v2.3. The reported results are the average of 3 experiments. We also set OMB to run each test 1,000 times for each message size.

A. Experimental Setup

We use the HPC Advisory Council High-Performance Center (HPCAC) [9] cluster for our evaluation. The Thor cluster at HPCAC has 32 nodes, each equipped with a BlueField-2 SmartNIC as well as a separate ConnectX-6 HCA. The SmartNIC adapters have an array of 8 ARM cores operating at 2 GHz and 16 GB of RAM. All BlueField-2 adapters are equipped with Mellanox MT41686 EDR ConnectX-6 HCAs (100 Gbps data rate) with PCIe Gen4 interfaces [8]. The Thor hosts are equipped with the Broadwell series of Xeon dual-socket, 16-core processors operating at 3.40 GHz with 256 GB RAM.

B. Performance of MPI_Iallgather

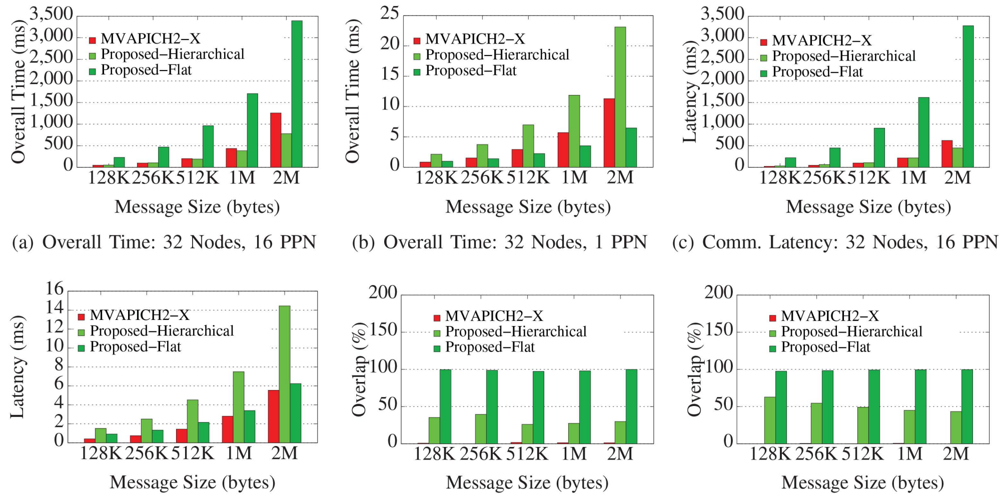
In this section, we compare the performance of MPI_Iallgather using the osu_iallgather benchmark from the OSU Micro Benchmark suite. Figure 4 indicates the communication latency of the benchmark, the overlap of communication and computation, and the overall execution time for each offload algorithm as well as the MVAPICH2-X algorithm for 32 node, 16 PPN and 32 node, 1 PPN jobs.

The OMB nonblocking benchmark suite first measures the amount of pure communication time by starting a timer, calling MPI_Iallgather, and then immediately calling MPI_Wait. Once the wait finishes, the timer is stopped. This value is the pure communication latency (i.e., Comm. Latency in the figures) without any computation.

The steps are repeated to measure overall execution time, however in between calling MPI_Iallgather and MPI_Wait, a dummy compute (matrix multiplication) is performed for the same amount of time as the pure communication latency. In cases where overlap is 100%, the time spent in MPI_Wait is thus close to zero, and computation is hidden. If there is 0% overlap–like in the CPU-based algorithms requiring the CPU–then almost all of the communication is performed during MPI_Wait. Since the compute time is roughly similar to the communication time, the benchmark calculates the overlap by using the formula $overlap = MAX(0, 100 - (((overall\_time - compute\_time)/pure\_comm\_time) * 100))$.

With these measurements, the subfigures show that the pure-host based algorithms are not able to provide any overlap. This is because the communication is progressed in MPI_Wait() by the CPU after the computation is finished. However, using the BlueField DPU to progress communication instead can provide 100% overlap using the Proposed Flat Algorithm and up to 62% using the Proposed Hierarchical Algorithm. The Proposed Flat Algorithm completely offloads communication to the DPU, allowing full overlap and retaining comparable communication latency to MVAPICH2-X for 32 nodes 1 PPN jobs. As a result, the Proposed Flat Algorithm achieves a 43% reduction in overall execution time.

However, for the 32 node 16 PPN results, it can be seen in Figure 4(c) that the flat algorithm actually does much worse in latency and overall time than MVAPICH2-X. The reason is that with large PPN counts, the shared memory channel is much faster than going over the network. Even with 100% overlap, if the communication latency is increased by a large enough amount, the overall time will increase. With this in mind, we choose the hierarchical algorithm for higher PPN jobs and the flat algorithm for lower PPN jobs in order to keep pure communication time comparable to the host, but retain as much overlap as possible since the overlap is the main source



(a) Overall Time: 32 Nodes, 16 PPN (b) Overall Time: 32 Nodes, 1 PPN (c) Comm. Latency: 32 Nodes, 16 PPN

(d) Comm. Latency: 32 Nodes, 1 PPN (e) Overlap: 32 Nodes, 16 PPN (f) Overlap: 32 Nodes, 1 PPN

Fig. 4. osu_iallgather Benchmark for the Proposed Allgather Algorithms Compared Against MVAPICH2-X

of benefit in our designs. The Proposed Hierarchical Algorithm for a 32 nodes 16 PPN job is able to reduce overall execution time by up to 38% compared to MVAPICH2-X.

C. Performance of MPI_Ibcast

Figure 5 shows the overlap of communication and computation, pure communication latency, and total execution time of the osu_ibcast benchmark for each offload algorithm as well as the MVAPICH2-X CPU-based broadcast. We measure the performance using the osu_ibcast benchmark test, which similarly to osu_iallgather, measures the nonblocking broadcast's pure communication latency and then posts another nonblocking broadcast, performs a matrix multiplication for the same amount of time as the communication latency, and then calls MPI_Wait.

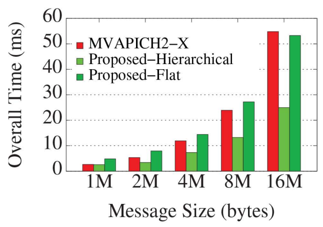
It is important to note that our DPU-based designs call the default MPI_Bcast in MVAPICH2-X and at the same time, the CPU-based baseline we compare to use similar algorithms for MPI_Ibcast (i.e., both MPI_Bcast and MPI_Ibcast use a scatter allgather for a large messages). Despite this, Figure 5(a) shows there is up to 54% reduction in total execution time of osu_ibcast for the 32 node 32 PPN case. The performance gain in this case comes from the fact that there is 100% overlap, but also the fact that the CPU-based results are doing a broadcast on all 1024 processes versus just those in the DPU communicator (1 DPU process per node–32x less processes). The read and writeback to host steps of the algorithm do not take long enough such that the overall time is increased.

VII. Related Work

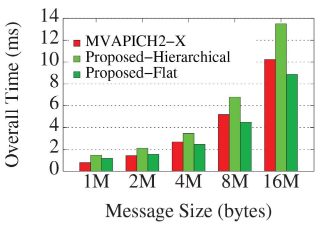
In this paper, we explore the use of Mellanox BlueField-2 DPU for offloading collective communication operations for nonblocking MPI_Iallgather and MPI_Ibcast. Previous research has been done for offloading collective operations using other hardware-based schemes. Authors in [5] implement a nonblocking MPI_Ialltoall on DPUs and demonstrate application-level benefits using a modified P3DFFT. Our paper extends this research and proposes new designs for nonblocking MPI_Iallgather and MPI_Ibcast. Authors in [10] design an offload mechanism for personalized collectives using RDMA primitives. For intra-node transfers, they use shared memory for small message transfers and RDMA loopback for large transfers to achieve overlap of communication and computation, as the HCA can progress the RDMA transfer in the background. The application of SmartNICs to the area of systems has been explored in various contexts. Floem [11] proposed a set of programming abstractions for the exploration of NIC-offloading designs. They demonstrate the efficacy of the system through key-value stores and distributed real-time data analytics applications. Liu et al. [12] propose a framework to offload distributed applications to SmartNICs using an actor model. Subramoni et al. [13] use the ConnectX-2 to create new communication primitives which can then be composed to implement collective operations with a focus on latency and overlap of communication and computation. Offloads for compute have been widely studied in the literature, such as with GPUs and coprocessors. MVAPICH2-MIC [14] offloads computation to the Intel Xeon Phi coprocessor.

VIII. Conclusion and Future Work

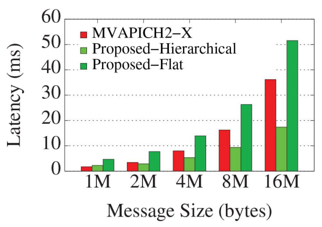
In this paper, we programmed the BlueField-2 DPU to offload several novel nonblocking broadcast and allgather algorithms using the MVAPICH2-X MPI library and compared each algorithm at the microbenchmark level. At the microbenchmark level, we have demonstrated a reduction in overall execution time of up to 43% for osu_iallgather and up to 54% for osu_ibcast. These results show that the BlueField-2



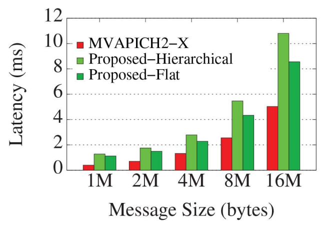
(a) Overall Time: 32 Nodes, 32 PPN



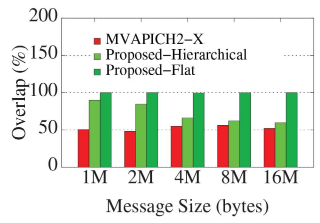
(b) Overall Time: 32 Nodes, 1 PPN



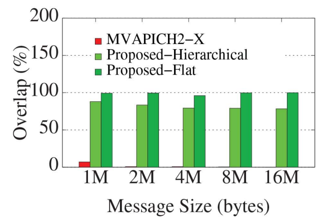
(c) Comm. Latency: 32 Nodes, 32 PPN



(d) Comm. Latency: 32 Nodes, 1 PPN



(e) Overlap: 32 Nodes, 32 PPN



(f) Overlap: 32 Nodes, 1 PPN

Fig. 5. osu_ibcast Benchmark for the Proposed Broadcast Algorithms Compared Against MVAPICH2-X

DPU can be an effective hardware offload mechanism for the MPI_Iallgather and MPI_Ibcast nonblocking collectives.

For our future work, we plan to investigate offloading other nonblocking collectives and study the benefits.

References

[1] "InfiniBand Trade Association," http://www.infinibandta.com, 2017.

[2] "The mvapich project: Transforming research into high-performance mpi library for hpc community," *Journal of Computational Science*, p. 101208, 2020.

[3] R. L. Graham, D. Bureddy, P. Lui, H. Rosenstock, G. Shainer, G. Bloch, D. Goldenerg, M. Dubman, S. Kotchubievsky, V. Koushnir *et al.*, "Scalable hierarchical aggregation protocol (sharp): a hardware architecture for efficient data reduction," in *2016 First International Workshop on Communication Optimizations in HPC (COMHPC)*. IEEE, 2016, pp. 1–10.

[4] Mellanox Corporation, "Understanding Tag Matching for Developers," https://community.mellanox.com/s/article/understanding-tag-matching-for-developers.

[5] M. Bayatpour, N. Sarkauskas, H. Subramoni, J. Hashmi, and D. Panda, "Bluesmpi: Efficient mpi non-blocking alltoall offloading designs on modern bluefield smart nics," June 2021.

[6] http://mvapich.cse.ohio-state.edu/benchmarks.

[7] S. Choi, M. Shahbaz, B. Prabhakar, and M. Rosenblum, "λ-NIC: Interactive Serverless Compute on SmartNICs," in *SIGCOMM Posters and Demos '19: Proceedings of the ACM SIGCOMM 2019 Conference Posters and Demos*, 2019, pp. 151–152. [Online]. Available: https://doi.org/10.1145/3342280.3342341

[8] "Mellanox BlueField ." [Online]. Available: https://docs.mellanox.com/x/iQO3

[9] "High-Performance Center Overview ." [Online]. Available: https://www.hpcadvisorycouncil.com/cluster_center.php

[10] H. Subramoni, A. A. Awan, K. Hamidouche, D. Pekurovsky, A. Venkatesh, S. Chakraborty, K. Tomko, and D. K. Panda, "Designing non-blocking personalized collectives with near perfect overlap for rdma-enabled clusters," in *High Performance Computing*, J. M. Kunkel and T. Ludwig, Eds. Cham: Springer International Publishing, 2015, pp. 434–453.

[11] P. M. Phothilimthana, M. Liu, A. Kaufmann, R. B. Simon Peter, and T. Anderson, "Floem: A Programming System for NIC-Accelerated Network Applications," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 13th USENIX Symposium on Operating Systems Design and Implementation, 2018.

[12] M. Liu, T. Cui, H. Schuh, A. Krishnamurthy, S. Peter, and K. Gupta, "iPipe: A Framework for Building Distributed Applications on SmartNICs," in *SIGCOMM '19: Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 318–333. [Online]. Available: https://doi.org/10.1145/3341302.3342079

[13] H. Subramoni, K. Kandalla, S. Sur, and D. K. Panda, "Design and evaluation of generalized collective communication primitives with overlap using connectx-2 offload engine," in *2010 18th IEEE Symposium on High Performance Interconnects*, Aug 2010, pp. 40–49.

[14] S. Potluri, K. Hamidouche, D. Bureddy, and D. K. Panda, "Mvapich2-mic: A high performance mpi library for xeon phi clusters with infiniband," in *2013 Extreme Scaling Workshop (xsw 2013)*, 2013, pp. 25–32.