Designing Hierarchical Multi-HCA Aware Allgather in MPI

Tu Tran
tran.839@osu.edu
The Ohio State University
Columbus, Ohio, USA

Benjamin Michalowicz
michalowicz.2@osu.edu
The Ohio State University
Columbus, Ohio, USA

Bharath Ramesh
ramesh.113@osu.edu
The Ohio State University
Columbus, Ohio, USA

Hari Subramoni
subramoni.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Aamir Shafi
shafi.16@osu.edu
The Ohio State University
Columbus, Ohio, USA

Dhabaleswar K. Panda
panda@cse.ohio-state.edu
The Ohio State University
Columbus, Ohio, USA

ABSTRACT

To accelerate the communication between nodes, supercomputers are now equipped with multiple network adapters per node, resulting in a "multi-rail" network. The second and third-placed systems of the Top500 use two adapters per node; recently, the ThetaGPU system at Argonne National Laboratory (ANL) uses eight adapters per node. With such an availability of networking resources, it is a non-trivial task to utilize all of them. The Message Passing Interface (MPI) is a dominant model for high-performance computing clusters. Not all MPI collectives utilize all resources, and this becomes more apparent with advances in bandwidth and adapter count in a given cluster.

In this work, we take up this task and propose hierarchical, multi-HCA aware Allgather designs; Allgather is a communication-intensive collective widely used in applications like matrix multiplication and other collectives. The proposed designs fully utilize all the available network adapters within a node and provides high overlap between inter-node and intra-node communication. At the micro-benchmark level, our new schemes achieve performance improvement for both single node and multiple node communication. We see inter-node improvements up to 62% and 61% better than HPC-X and MVAPICH2-X for 1024 processes. The design for inter-node communication also boosts the performance of Ring Allreduce by 56% and 44% compared to HPC-X and MVAPICH2-X. At the application level, the enhanced Allgather shows 1.98x and 1.42x improvement in a matrix-vector multiplication kernel when compared to HPC-X and MVAPICH2-X, and Allreduce performs up to 7.83% better in deep learning training against MVAPICH2-X.

KEYWORDS

MPI, Collectives, Network-aware, HCA-aware, Allgather, Allreduce

ACM Reference Format:
Tu Tran, Benjamin Michalowicz, Bharath Ramesh, Hari Subramoni, Aamir Shafi, and Dhabaleswar K. Panda. 2022. Designing Hierarchical Multi-HCA Aware Allgather in MPI. In *51th International Conference on Parallel Processing Workshop (ICPP Workshops '22), August 29-September 1, 2022, Bordeaux, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3547276.3548524

ICPP Workshops '22, August 29-September 1, 2022, Bordeaux, France

ACM ISBN 978-1-4503-9445-1/22/08…$15.00
https://doi.org/10.1145/3547276.3548524

1 INTRODUCTION

Exascale computing is the next milestone after the presence of many Petascale supercomputers, coming with the computing power of 10^{18} floating-point operations per second (FLOPS). The Exascale Computing Project (ECP) [21] was initiated as a primary attempt to pursue Exascale computing by the US government in 2016 due to its huge potential of contributing to the American society and the world through scientific discovery, energy assurance, economic competitiveness, national security, pandemic simulations such as COVID, etc. Consequently, individual compute nodes are getting more and more powerful, with some being equipped with over 100 CPU cores, fast/high-throughput memory, and multiple accelerators such as GPUs. Additionally, many compute nodes are connected by low latency, high bandwidth interconnects such as InfiniBand. To further increase bandwidth between compute nodes, supercomputers are now equipped with more than one network adapter per node, resulting in a multi-rail network. Examples of such a system are Frontier [11] and El Capitan [10], the first exascale computers planned to debut in 2022 and 2023. With enormous computing capabilities, it is not a trivial task to fully harness such systems to their fullest extent.

Among commonly used parallel programming models such as shared memory, message passing, and partitioned global address space (PGAS) [7, 40], the Message Passing Interface (MPI) standard [20] is currently the de-facto parallel programming model on modern high-performance computing (HPC) clusters. In 2018, Bernholdt et al. [4] surveyed the MPI usage in the ECP. Out of the 97 projects active at the time of the survey, 77 responses were received, and 56 reported they were using MPI. In preparation for Exascale, MPI has been continuously studied, analyzed, and improved ever since 2009 by the MPI community [2, 34]. The MPI standard defines two main types of communication: point-to-point and collective. The latter is communication-intensive, involves two or more processes in a communicator, and contributes a significant part of the total runtime of many HPC applications.

1.1 Motivation

In the Top500 [37] (accessed in November 2021), the current second and third largest systems, namely Summit and Sierra, utilize

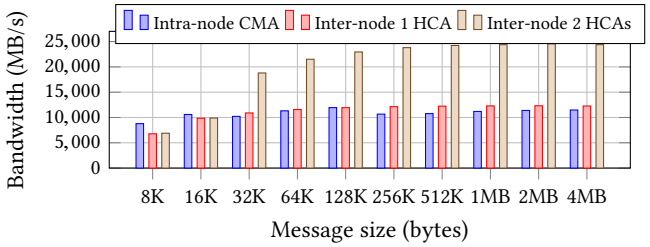


Figure 1: Bandwidth comparison between intra-node and inter-node communication

two adapters per node, and recently the ThetaGPU system [36] at Argonne national laboratory equips eight adapters per node to accelerate inter-node communication. Although HPC clusters are getting more network resources for inter-node communication by simply equipping more adapters, intra-node bandwidth roughly stays the same unless a cluster is replaced by another CPU architecture with a faster memory subsystem. Figure 1 shows the bandwidth differences between intra-node communication using CMA (Cross Memory Attach) and inter-node communication using one and two HCAs at MPI-level between two processes. As shown here, the bandwidth of inter-node communication with one HCA is approximately equal to that of intra-node communication, with inter-node bandwidth doubling when a second HCA is utilized. The details of the experimental environment of all of the experiments in this paper can be found in Section 5. As a result, existing collective designs need to be revisited and augmented to be able to fully utilize the additional network resources.

Broadly speaking, there are two categories of collective algorithms: flat and two-level. Conventional flat algorithms [35], such as Ring, do not differentiate intra-node and inter-node communication. For example, large messages inside an Allgather with more than one process per node (PPN) are collected in a ring fashion. In each communication step, a node sends data to its right peer and receives it from its left; If there are N processes participating in an Allgather routine, there will be $N-1$ communication steps. Therefore, the communication will be bottlenecked by the slowest links—intra-node transfers in this case. To further demonstrate the issue, we performed an experiment with 2 nodes and 2 PPN performing an Allgather in a ring fashion on a cluster with 2 adapters per node. Figure 2 shows the timeline view of communication events extracted from Tuning and Analysis Utilities (TAU) [32] and redrawn. The tracing capability from TAU allows us to present when/where events happened along a global timeline as well as when/where messages were sent. Another collective that employs a ring algorithm for large messages is Allreduce. This collective has been heavily studied and is still being continuously improved by the academic community [3, 5, 12]; it is not only intensively used in traditional HPC applications but is commonplace in the deep learning (DL) world.

The second category is two-level [14], also referred to as single-leader or multi-leader based designs. Within a node, processes are divided into one or multiple groups, and each group contains a designated leader. In the first phase, all processes share data with group leaders. In the second phase, leaders from each node perform a data exchange using a flat algorithm. Finally, The leaders

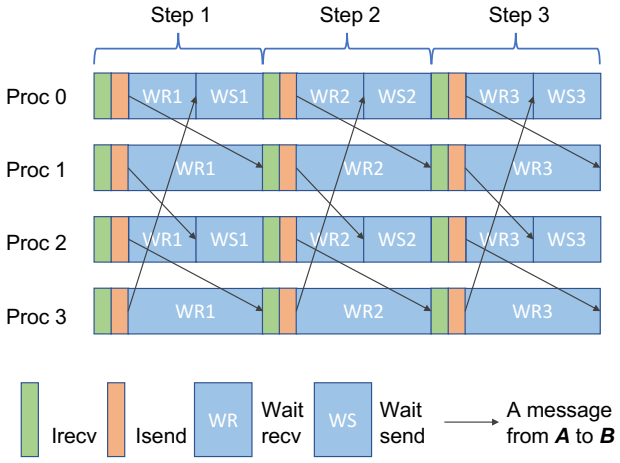


Figure 2: Allgather 2 Nodes, 2 PPN communication timeline visualization

broadcast the result to their intra-node peers. In the multi-leader based design proposed by Kandalla et al. [14], the communication in the second phase is a blend of data exchanges between leaders within and across nodes using conventional flat algorithms like Ring; this can potentially lead to a bottleneck due to the difference in intra-node and inter-node bandwidth. The bottleneck is clearly demonstrated in Figure 2. The problem will only get worse for multi-rail networks; for instance, in the system we are measuring, inter-node bandwidth is roughly twice that of intra-node. Besides that, the authors clearly separated the communication phases. A phase starts right after the previous one has finished while phases two (inter-leader data exchange) and three (node-level data distribution) can be overlapped. The work of Mamidala et al. [19] can be considered as a single-leader design with overlap of inter-node and intra-node communication. The design overcomes the two aforementioned drawbacks of the multi-leader design. However, for inter-leader data exchanges in phase two, only Recursive Doubling is used. In this paper, we show that Ring can also be adopted in this case for more overlap of network operations and intra-node shared memory copies.

As a result, we ask the following questions:

- **For pure intra-node communication, can we reduce the latency by utilizing idle network adapters? If so, how can we efficiently distribute the communication workload between processes and adapters?**
- **For inter-node communication with multiple processes per node, can existing designs deliver good performance on multi-rail networks? If not, what are the sources of bottlenecks and how can we resolve them?**

1.2 Contributions

In this work, we propose a scalable and hierarchical multi-rail-aware Allgather. This collective is a communication-intensive and widely used operation in which every process broadcasts its data (AlltoAll Broadcast). Allgather is an important collective used in many applications such as lower and upper triangle factorization, solving differential equations, linear algebra operations such as

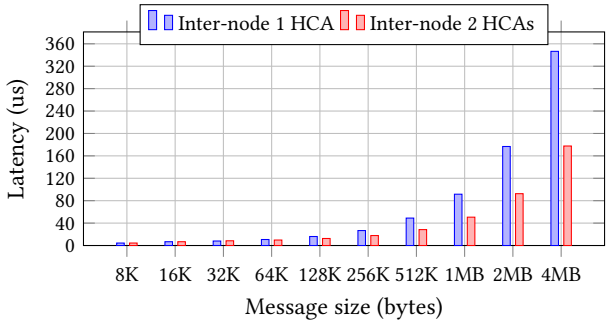


Figure 3: Inter-node Latency Comparison on two processes with One and Two HCAs

Bayesian Probabilistic Matrix Factorization [29, 39, 41], and matrix-matrix/matrix-vector multiplication [8, 41]. The designs not only fully utilize all of the available network adapters in a node but also provide high overlap between inter-node and intra-node communication.

At the micro-benchmark level, our new schemes achieve up to 62% improvement against HPC-X and 61% against MVAPICH2-X. Furthermore, the design for inter-node communication boosts the performance of Ring Allreduce by 56% and 44% compared to HPC-X and MVAPICH2-X. At the application level, the enhanced Allgather shows 1.98x and 1.42x improvement in a matrix-vector multiplication kernel when compared to HPC-X and MVAPICH2-X, and Allreduce performs up to 7.83% better in deep learning training against MVAPICH2-X.

The rest of the paper is organized as follows. Section 2 gives the background needed for this paper. In Section 3, we present our multi-HCA aware schemes for Allgather. The performance models of the the proposed designs are discussed in Section 4. Section 5 details the performance evaluation results for Allgather and shows how it can improve Allreduce at microbenchmark and application levels. Research work related to ours is presented in Section 6. Our conclusions and thoughts for future work are presented in Section 7.

2 BACKGROUND

2.1 Multi-rail point-to-point level design

In 2004, Liu et al. [17] proposed a point-to-point design at MPI level for Multirail InfiniBand clusters. This is one of the first works that tackled the use of multiple network adapters to accelerate inter-node communication. For small messages, messages are sent through different rails in a round-robin fashion. This approach is good for load balancing between different rails. As the message size increase, the bandwidth usage of a rail will go up and eventually get saturated. A technique called striping will be used in this case to overcome and lessen the bandwidth bottleneck. To clarify, messages are broken into many chunks and sent across multiple rails simultaneously. Figure 1 and 3 illustrate the bandwidth and latency tests between to processes on two nodes at MPI level using OSU micro-benchmark [22]. At 16 KB, the bandwidth of a rail is saturated, and the striping technique applies to any messages with size greater than this. As a result, the latency for large messages is reduced by half.

2.2 Conventional flat Allgather designs

There are many conventional algorithms for Allgather, with the most popular ones being Recursive Doubling (RD), Bruck's, Ring, and Direct Spread. In this paper, we focus on RD, Ring and Direct Spread.

(1) Recursive Doubling (RD) executes in $log_2(N)$ steps, where N is the number of processes. In step i, the distance between any communicating processes is 2^{i-1}, and the message size increases by a factor of two after each step. For non-power-of-two processes, RD requires additional steps to complete the communication.
(2) In the Ring algorithm, data are exchanged along a virtual ring of processes. In each step i, a process rank r sends the data it has received in the previous step to its right peer (rank $r + 1$) and receives from its left peer (rank $r + 1$). If there are N processes participating in the communication, the algorithm will take $N - 1$ steps to finish.
(3) In Direct Spread (dissemination) Allgather algorithm, each process directly communicates with the source processes to get the data instead of getting from through an intermediate process the way Ring algorithm does. Specially, in step i, a process rank r receives data directly from the process rank $(r - i)\%N$. This algorithm requires $N - 1$ steps to complete like Ring for a communication of N processes

In summary, these algorithms were designed and optimized for systems with a single processor or single communicating process per node with homogeneous communication channels in terms of latency and bandwidth. Recursive Doubling is good for short messages. Ring is better than Direct Spread due to the nearest-neighbor communication pattern even though Ring has data dependencies on other processes, and Direct Spread does not. They are both well-suited for long messages.

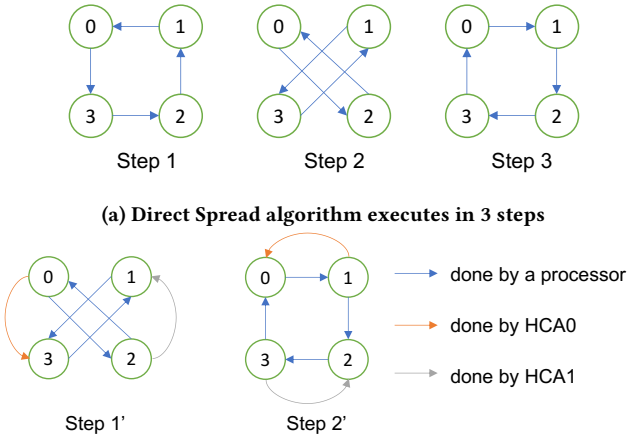
2.3 Intra-node Communication Mechanisms

For intra-node communication, in the context of collective communication, processes can exchange data by directly using point-to-point operations like Send/Recv. However, this approach incurs overheads caused by tag matching, handshaking via the rendezvous protocol, etc.

Another approach is to use shared memory, which can eliminate unnecessary overheads in the previous approach. However, two copies are required for this method, one copy from the sender's buffer to the shared buffer and one copy from the shared buffer to the recipient's buffer, which results in a significant performance degradation for large messages (usually >= $16KB$). To avoid the additional copy, kernel-assisted single-copies mechanisms such as LiMiC [9], KNEM [18], and Cross Memory Attach (CMA) [6] are deployed. While LiMiC and KNEM require some installations as kernel modules, CMA is integrated inside newer versions of Linux kernel and available to use.

2.4 Ring Allreduce

Allreduce is widely used in traditional HPC applications [16] and emerging deep learning frameworks such as CNTK [30], Tensorflow [1], and Horovod [31]. In 2019, Laguna et al. showed that it is mostly used among MPI collectives [16]. Among the proposed algorithms



(a) Direct Spread algorithm executes in 3 steps

(b) The proposed MHA-intra executes in 2 steps. Step 1' is step 2 overlapping with 1st half of step 1. Step 2' is step 3 overlapping with 2nd half of step 1.

Figure 4: 4 processes performing Allgather: MHA-intra Allgather with the aid of 2 HCAs comparing with Direct Spread

[3, 5, 12] and many others, Ring Allreduce proposed by Patarasuk et al. [27] is proven to be bandwidth-optimal, which is particularly suitable for large messages: all processes exchange data in a logical ring manner, and the algorithm executes in two phases. In the Reduce-Scatter phase, assuming there are N participating processes, a logical ring communication is performed in $N-1$ iterations. After that, an Allgather is performed, and as a result, each process will have a complete reduction vector.

3 THE PROPOSED DESIGNS

3.1 A multi-HCA aware design for intra-node communication

Allgather is a symmetric collective in which every process does the same amount of work; to reduce the communication time, every process needs to reduce the time it takes to complete its workload. The proposed design is extended from the Direct Spread algorithm mentioned in Section 2.2. Figure 4a illustrates the algorithm with four processes participating in the communication. With idling network adapters, each process can further accelerate the communication by assigning a fraction of their workload to them. Figure 4b demonstrates how using two available network adapters can finish the communication in two steps instead of three, assuming a processor and an adapter take roughly the same time to transfer a message.

If each process assigns too much or too little workload to adapters, the communication will be hampered by the component taking the longest time to finish its task, either from processors or adapters. As a result, we must make sure that appropriate amounts of workload are handled by each process and adapter so that they can finish roughly at the same time. Here a tuning algorithm is proposed to find an optimal workload to offload to adapters. Figure 5 shows the relationship between the offload size to adapters and the time it takes to finish the communication. Based on the correlation, we can easily find the optimal point by first measuring the time if

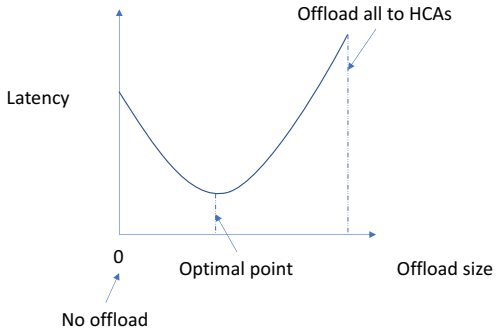


Figure 5: A chart showing the correlation between the offload size to adapters and latency

all of the communication is done by adapters and processors stay idle. After that, the offloaded workload is gradually decreased until the intersection of the downward and upward trend lines of the communication latency is found.

3.2 A hierarchical multi-HCA aware design for inter-node and intra-node communication

The details of the proposed design for inter-node communication with multiple processes per node, namely a hierarchical multi-HCA aware design are as follows:

- Phase 1: Node-level data aggregation using the proposed intra-node allgather in Section 3.1.
- Phase 2: Data transfers between group leaders with a single process leader per node using either RD or Ring, depending on message sizes. All processes within a node form a group, and each group has a group leader.
- Phase 3: Node-level data distribution with group leaders copying to shared memory and group members copying out.

To further elaborate, during the intra-node communication, usually done with processors performing memory copies, in phase 1, network adapters are idle, which leads to inefficient resource utilization. The proposed intra-node allgather is used to have better utilization of network resources. In phase 2, inter-leader data exchanges of N nodes can be done in $\log N$ steps with RD or $(N-1)$ steps with Ring. Node-level data distribution in phase 3 can be overlapped with phase 2 by using shared memory. As soon as, a data chunk arrives in each step in phase 2, group leaders can copy it into shared memory, and then increase a counter that indicates the availability of a data chunk in each step. Non-leader processes check the counter for the arrived chunk to copy out into their buffers. As a result, network transfers and intra-node memory copies can be overlapped, which is demonstrated in Figure 6.

For inter-leader data exchanges in phase 2, Ring can perform better and deliver more overlap than RD, depending on message size. Figure 7 depicts the case of 8 leaders corresponding to 8 nodes with Ring outperforming RD due to higher overlap. For RD, The size of data transferred in a current step is twice the one in the previous step. This is why RD loses its overlapping capability. Specifically, inter-node transfer of size D happens concurrently with intra-node broadcast of size $(D/2)$ instead of size D as in the case of Ring. In addition, after the final data chunk has arrived, leader processes

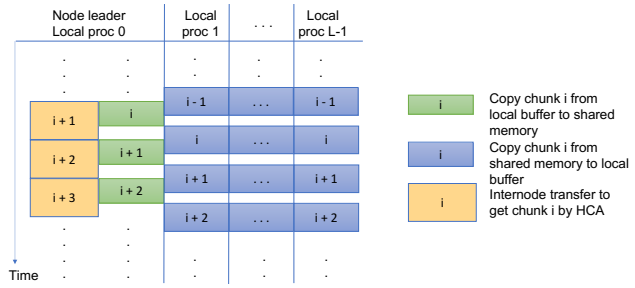


Figure 6: A timeline view of communication events of a node during interleader data exchange and node-level data distribution phases

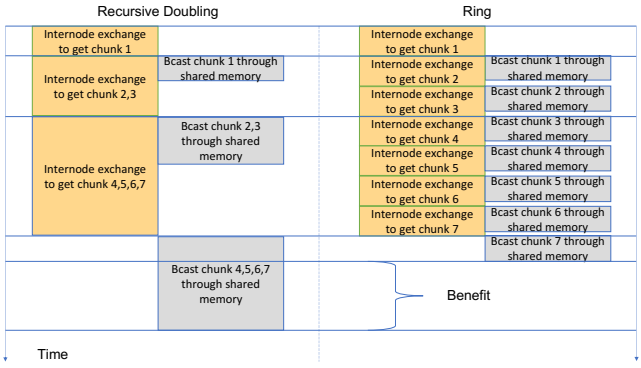


Figure 7: A comparison of Recursive Doubling and Ring algorithms used in inter-leader data exchange phase

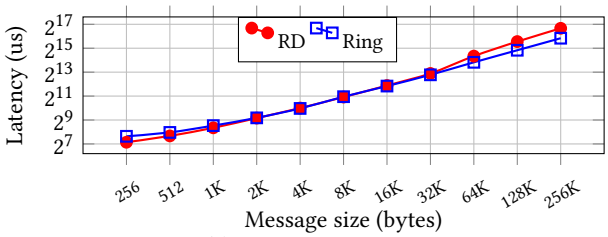
need to do one final broadcast of that chunk; the data size of the final chunk of RD is $2^{(log_2(N)-1)}$ times bigger than the one of Ring. As the number of nodes goes higher, we see a better level of overlap delivered by Ring when compared to RD. Figure 8 compares the performance of Ring and RD used in the inter-leader data exchange phase. We see that RD outperforms Ring for small message sizes. Note that the message size shown in the figures is the message size of each process contributing to the Allgather; the real transferred message size by node leaders is PPN times bigger than this.

4 PERFORMANCE MODELS OF THE MULTI-HCA AWARE DESIGNS

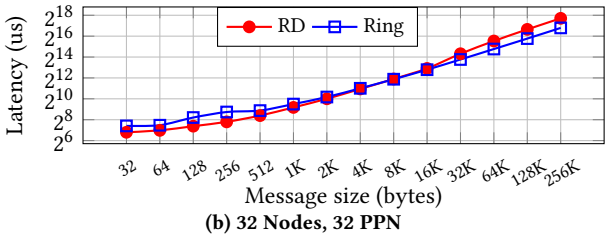
4.1 Modeling the Cost of MHA-intra Allgather

Assuming there are L processes participating in an Allgather, in the MHA-intra algorithm, they first copy their data from send to receive buffers if the operation is not an in-place operation. After that, each process requests H HCAs to do d transfers, while it does $(L-1-d)$ intra-node transfers. d is the optimal number of offloaded transfers to HCAs per process, depending on the number of processes L and message size M; For optimal communication, we need to distribute the workload from processes to HCAs so that they can approximately finish at the same time. As a result, the following equation can be used to find d:

$$\begin{aligned} &T_C(M) * (L-1-d) = T_H(M) * L * d \\ &\Rightarrow d = (T_C(M) * (L-1))/(T_H(M) * L + T_C(M)) \end{aligned} \quad (1)$$



(a) 16 Nodes, 32 PPN



(b) 32 Nodes, 32 PPN

Figure 8: Comparison of the RD and Ring Algorithms in the Proposed Design During Inter-Leader Data Exchange

Table 1: Notations used in the cost models

Symbol	Description
N	Number of nodes
L	Number of processes per node
M	Message size
H	Number of adapters
α_C	Startup time per intra-node transfer
BW_C	Bandwidth of intra-node transfer
α_H	Startup time per inter-node transfer
BW_H	Bandwidth of inter-node transfer
α_L	Startup cost per local memory copy
BW_L	Bandwidth of local memoy copy
$T_C(M)$	Time to send an intra-node message of size M
$T_H(M)$	Time to send a message of size M using H adapters
$T_L(M)$	Time to perform a memory copy of size M

In addition, a transfer of message M by H adapters can be modeled as $T_H(M) = \alpha_H + M/(BW_H * H)$. A local memory copy of size M can be modeled as $T_L(M) = \alpha_L + M/BW_L$. An intra-node transfer of message M can be modeled as $T_C(M) = \alpha_C + (M/BW_C) * b$, in which b is a number of concurrent accesses to memory. It is used to model the congestion when memory bandwidth is saturated with large messages. For small messages, b has a value of one.

As a result, the MHA-intra Allgather can be estimated as follows:

$$\begin{aligned} T_{MHA-intra}(M) = &\ T_L(M) \\ &+ Max\{\ (L-1-d) * T_C(M), \\ &\quad L * d * T_H(M)\ \} \end{aligned} \quad (2)$$

4.2 Modeling the Cost of MHA-inter Allgather

For MHA-inter Allgather, the communication happens in 3 phases. In phase 1, data are shared with group leaders using MHA-intra

algorithm, then the cost is $T_{MHA-intra}(M)$, modeled in the previous section. For phase 2, group leaders perform data exchange of size $(M * L)$, either using RD or Ring. While RD runs in $\log N$ steps with data size doubled in every step, Ring executes in $N-1$ steps with data size of $(M * L)$. As a result, the cost for phase 2 is

$$\begin{aligned} T_{phase2-RD}(ML) &= T_{step\,1} + T_{step\,2} + \ldots + T_{step\,log(N)} \\ &= T_H(M*L) + T_H(2*M*L) \\ &\quad + \ldots + T_H(log(N)*M*L) \\ &= \alpha_H * log(N) + (N-1)*(M*L)/(BW_H * H) \end{aligned} \tag{3}$$

$$\begin{aligned} T_{phase-2-Ring}(ML) &= T_{step\,1} + T_{step\,2} + \ldots + T_{step\,(N-1)} \\ &= T_H(M*L) + T_H(M*L) \\ &\quad + \ldots + T_H(M*L) \\ &= \alpha_H * (N-1) \\ &\quad + (N-1)*(M*L)/(BW_H * H) \end{aligned} \tag{4}$$

For the data distribution of node leaders in phase 3, the leaders perform multiple broadcasts of size $(M * L)$ by first copying to shared memory; then, its peers can copy out to their local buffers. When copying out, all the peers cannot do it concurrently because of memory congestion. As a result, the cost of copying out of $(L-1)$ processes is the cost of memory copy of one process times the congestion factor $cg(M, L-1)$, which is a function of $(L-1)$ processes accessing a shared region of M bytes and thus can be empirically measured. Consequently, a broadcast can be modeled as

$$\begin{aligned} &T_{intra\,bcast}(M*L) \\ &\quad = T_{copy\,in}(M*L) + T_{copy\,out}(M*L) \\ &\quad = (\alpha_L + (M*L)/BW_L) \\ &\quad\quad +(\alpha_L + (M*L)/BW_L) * cg(M*L, L-1) \end{aligned} \tag{5}$$

When phase 2 overlaps with phase 3, an inter-node transfer to get chunk i + 1 happens concurrently with an intra-node broadcast of chunk i. Phase 2 ends when leaders receive the last chunk; then, they can do a final broadcast to complete the communication. As a result, MHA-inter Allgather can be modeled as follows:

$$\begin{aligned} &T_{MHA-inter-RD}(M) \\ &\quad = T_{phase-1} + T_{phase-2} + T_{intra\,bcast}(M*L*N/2) \\ &\quad\quad \text{, if } T_{intra\,bcast}(M*L) <= T_H(2*M*L) \\ &\quad = T_H(M*L) + (N-1) * T_{intra\,bcast}(M*L), \\ &\quad\quad \text{, otherwise} \end{aligned} \tag{6}$$

$$\begin{aligned} &T_{MHA-inter-Ring}(M) \\ &\quad = T_{phase\,1} + T_{phase\,2} + T_{intra\,bcast}(M*L) \\ &\quad\quad \text{, if } T_{intra\,bcast}(M*L) <= T_H(M*L) \\ &\quad = T_H(M*L) + (N-1) * T_{intra\,bcast}(M*L), \\ &\quad\quad \text{, otherwise} \end{aligned} \tag{7}$$

4.3 Model Validation

To predict the performance of MHA-intra and MHA-inter Allgather, we must first empirically obtain parameters in Table 1. For intra-node communication, Equation (2) is used to estimate the cost of MHA-intra. Figure 9 shows that the predicted latency is close to

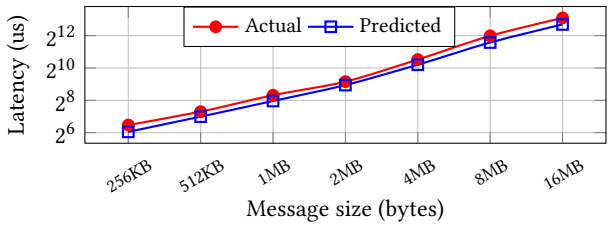


Figure 9: Validation of MHA-intra with 4 processes

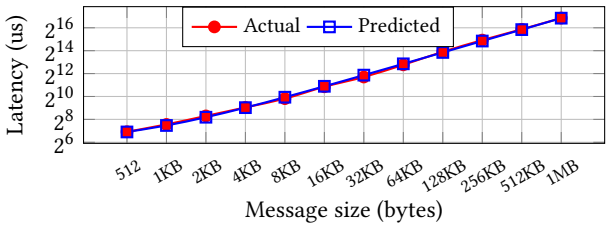


Figure 10: Validation of MHA-inter with 8 nodes 32 PPN

the actual latency of MHA-intra, which means the proposed model can estimate the trend properly. For inter-node communication, equations (6) and (7) can be used to estimate the cost of MHA-inter when the algorithm for inter-leader data exchange is RD and Ring, respectively. In Figure 10, the predicted latency and the actual latency reflect the tuned algorithm used in phase two between RD and Ring. We can see that the estimated numbers from the proposed model are are comparable with the measured numbers. As a result, by using the two models, we can predict how much performance can be improved for a communication pattern of N nodes with L PPN on a system of H adapters.

5 PERFORMANCE EVALUATION

In this section, we first present the environment for evaluation and then provide the results of experiments performed to evaluate the performance of the proposed designs at the microbenchmark and application levels.

5.1 Experimental environment

All of the experiments presented in this paper are conducted on Thor cluster of HPC Advisory Council [13]. It consists of 32 nodes equipped with dual-socket Intel® Xeon® 16-core CPUs E5-2697A V4 @ 2.60 GHz (Broadwell), 1024 cores in total. Each node is equipped with 2 ConnectX-6 HDR100 100Gb/s InfiniBand adapters and 256GB DDR4 2400MHz RDIMMs. The operating system used is Rocky Linux 8.5 (Green Obsidian), with kernel version 4.18.0-348.12.2.el8_5.x86_64 and Mellanox OFED version 5.5-1.0.3.2.

On the software aspect, the proposed designs are compared with two widely used MPI implementations in the scientific community, namely MVAPICH2-X version 2.3 [23] and HPC-X version 2.10.0 [24]. MVAPICH2 delivers the best performance, scalability and fault tolerance for high-end computing systems and servers using InfiniBand, Omni-Path, Ethernet/iWARP, RoCE, Cray Slingshot 10, and Rockport Networks networking technologies. NVIDIA® HPCX® is a variant of Open MPI [25] maintained by NVIDIA, which provides high performance, scalability, and efficiency and ensures that

communication is fully optimized for NVIDIA InfiniBand networking solutions. Additionally, for the performance evaluation of deep learning, we use PyTorch version 1.8.0 [26] and Horovod version 0.20.0 [31]. Finally, each of the numbers reported here is an average of at least three runs. Each run has 1000 warm-up and measurement iterations.

5.2 Intra-node Allgather evaluation

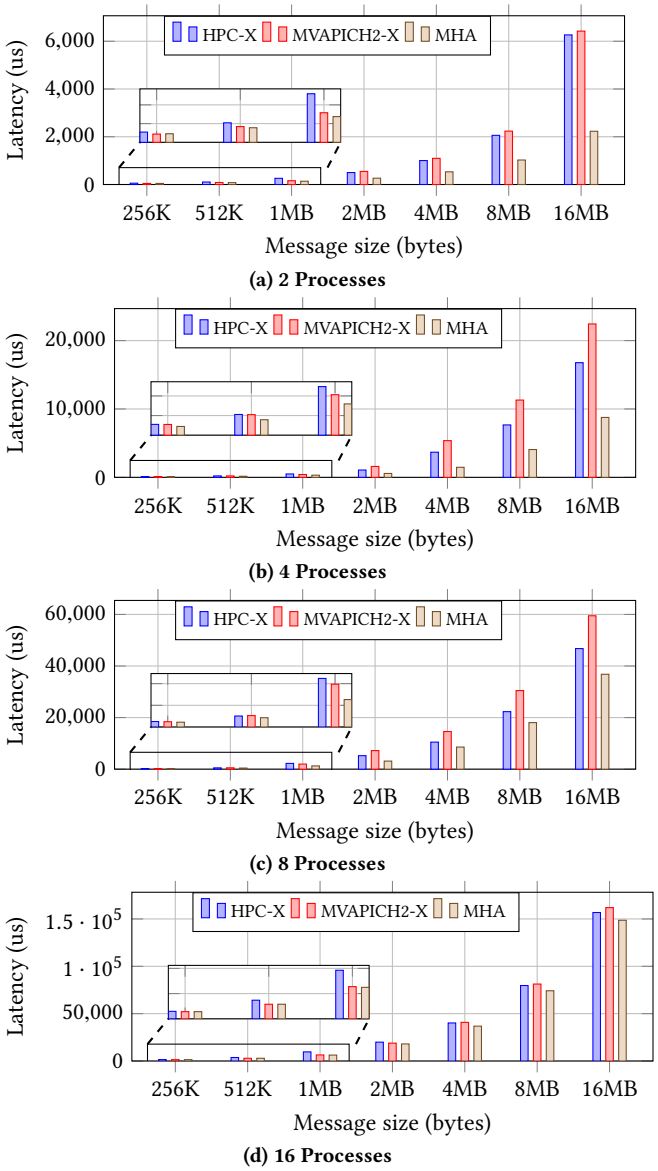
Figure 11a, 11b, 11c, and 11d show the performance evaluation of Allgather with different numbers of processes participating in the communication using OSU microbenchmark [22]. The proposed design with the assistance of 2 available HCAs, when compared to HPC-X and MVAPICH2-X, speeds up the performance up to 64% and 65% for two processes, 60% and 73% for four processes, 44% and 56% for eight processes, and 35% and 10% for 16 processes, respectively. The numbers with 32 processes are not included here because the performance improvement is less than 10%. One thing to note: as the number of processes in the communication increases given a constant number of adapters, the performance benefit decreases, which is an expected trend. To reduce the communication time, each process offloads a portion of its workload to HCAs with the objective that processes and HCAs can finish at the same time. As more processes participate in the communication, the offloaded portion gets smaller because the HCAs also have to process the workload from the additional processes. The offloaded portion represents the reduction in communication latency of each process. A smaller portion means less performance improvement. As a result, more adapters are needed for sustained performance when more processes are involved in the communication.

5.3 Inter-node Allgather evaluation

For inter-node communication, we perform experiments with 8, 16, and 32 nodes. Figures 12, 13, and 14 compare the performance of the proposed designs with MVAPICH2-X and HPC-X when running with 256, 512 and 1024 processes, respectively. When compared to HPC-X and MVAPICH2-X, the proposed design shows up to 29% and 21% better for 256 processes, 44% and 53% better for 512 processes, and 62% and 61% better for 1024 processes, respectively. As the number of nodes increases, the performance of the proposed design is also enhanced. By decoupling inter-node and intra-node communication with a single leader per node in the proposed design, multiple HCAs are utilized efficiently for communication across nodes. The second factor that contributes to the gain in performance comes from a higher overlap provided by Ring than RD during the inter-node distribution phase. The numbers shown are tuned numbers between these two algorithms. The proposed designs also show improvement for different numbers of processes per node; due to space limits, the results are not shown here.

5.4 Accelerating Allreduce with MHA Allgather

Allgather is used by several collectives, and Allreduce is one of them. In Ring-Allreduce, a reduce-scatter is first performed and then followed by an Allgather. As a result, by improving Allgather, the performance of Allreduce is also enhanced. Figure 15 depicts



(a) 2 Processes

(b) 4 Processes

(c) 8 Processes

(d) 16 Processes

Figure 11: Evaluation of Proposed Intra-node MPI_Allgather Design against state of the art libraries via OSU Microbenchmarks

the performance of the improved Allreduce compared to HPC-X and MVAPICH2-X. The improved Allreduce performs up 34% and 15% better for 256 processes, 39% and 31% better for 512 processes, and 56% and 44% better for 1024 processes than HPC-X and MVAPICH2-X, respectively. We can see that the proposed Allgather helps Allreduce scale better as the number of processes increases than the other two compared MPI implementations.

5.5 Impact of MHA Allgather on Matrix-Vector Multiplication

Allgather is not only used in other MPI collectives, but also in many applications such as lower and upper triangle factorization, solving differential equations, basic linear algebra operations such as

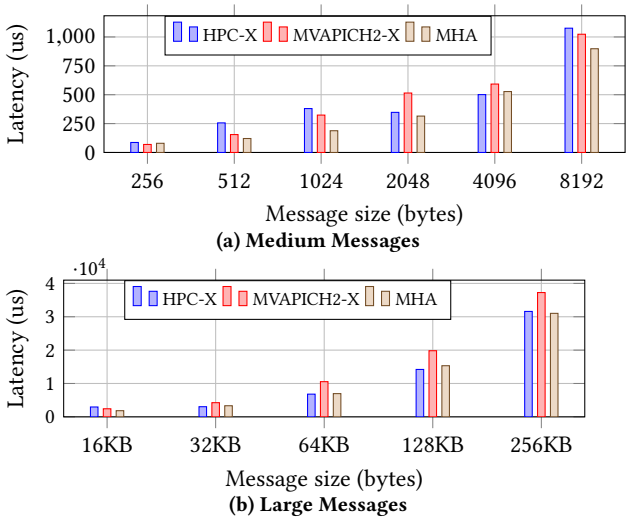


Figure 12: Proposed MPI_Allgather against state of the art libraries via OSU Microbenchmarks on 256 processes (8 nodes 32 PPN)

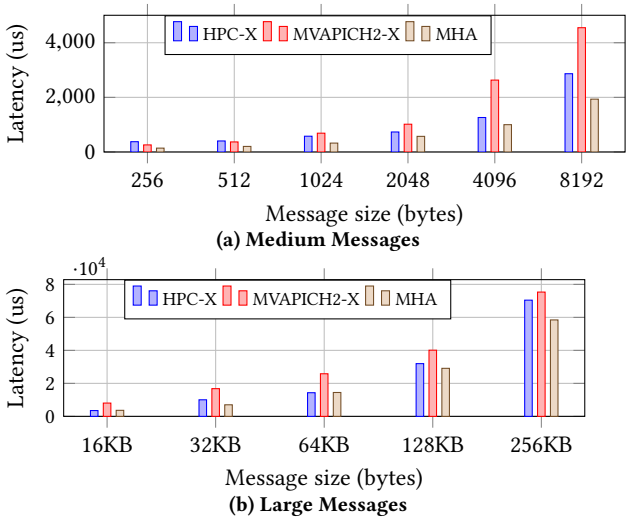


Figure 13: Proposed MPI_Allgather against state of the art libraries via OSU Microbenchmarks on 512 processes (16 nodes 32 PPN)

Bayesian Probabilistic Matrix Factorization [29, 39, 41], and matrix-matrix or matrix-vector multiplication [8, 41]. In this paper, to demonstrate the performance of the proposed Allgather at the application level, we evaluate matrix-vector multiplication $y = A*x$ in which A is a matrix of size M by N, X any Y are input and output vectors of size N by 1 and M by 1, respectively. A is partitioned using 1D row layout, in which each process holds $(M/number\ of\ processes)$ rows. Similarly, vector x any y are broken into equal segments of size $(N/number\ of\ processes)$ and $(M/number\ of\ processes)$ stored by each process. To do matrix-vector multiplication, each process first broadcasts the input segment it stores, resulting in an Allgather (All-to-all Broadcast); after that, they perform the multiplication locally to create their corresponding output segments. Figure 16 demonstrates the performance of the matrix-vector multiplication kernel in GFLOP/s (higher is better). In these experiments, we configure the problem size (M by N) so that communication contributes

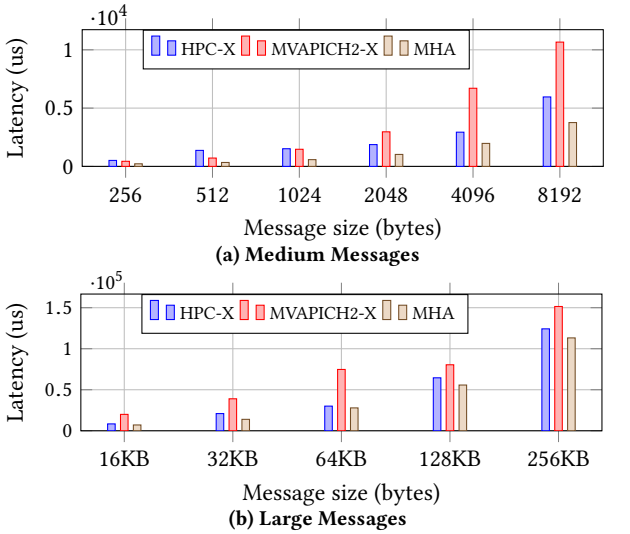


Figure 14: Proposed MPI_Allgather against state of the art libraries via OSU Microbenchmarks on 1024 processes (32 nodes 32 PPN)

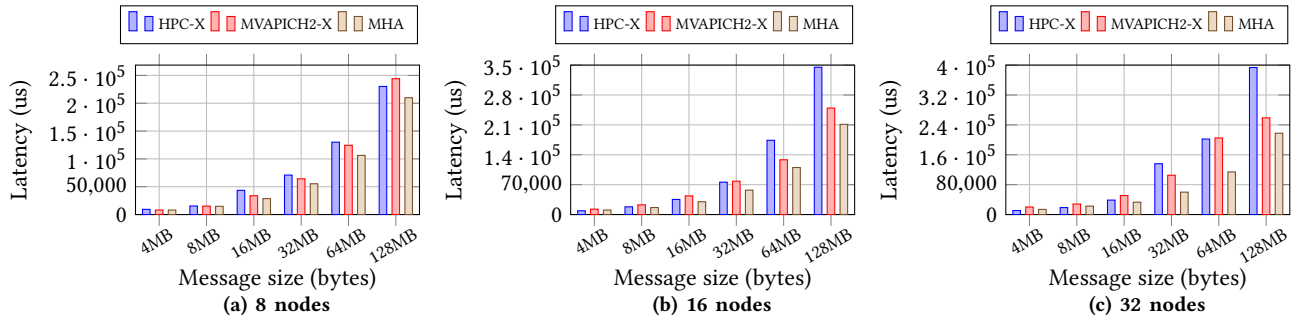
a significant time in the total runtime of the kernel to see the impact of the improved Allgather; In other words, the matrix A and input vector are long. The proposed Allgather outperforms both HPC-X and MVAPICH2-X by up to 1.98x and 1.42x for strong scaling and 1.84x and 1.94x for weak scaling experiments with 1024 processes.

5.6 Impact of the improved Allreduce on Deep Learning training

To quantify the benefit of the improved Allreduce at the level, we compare performance of training different neural networks in the increasing order of the number of parameters using PyTorch and Horovod. To be specific, we run the synthetic benchmark provided by Horovod with a batch size of 16. This is the largest batch size that the evaluated cluster can run without running out of memory. The three neural networks are ResNet50, ResNet101, and ResNet152 with 25.6, 44.7 and 60.4 millions of parameters, respectively [15]. Due to technical issues, we cannot set up HPC-X to work with Pytorch + Horovod despite our best effort. Open MPI cannot work with several versions of Horovod, reported in Horovod's website. Because HPC-X is optimized based on Open MPI, this may be the reason. Figure 17 shows that as the number of processes increases, we observe better performance, up to 7.83% better than MVAPICH2-X in both epoch time and images per second for ResNet50. In addition, when switching to a larger neural network (ResNet101 or ResNet152), we see similar performance benefits when running at similar scales.

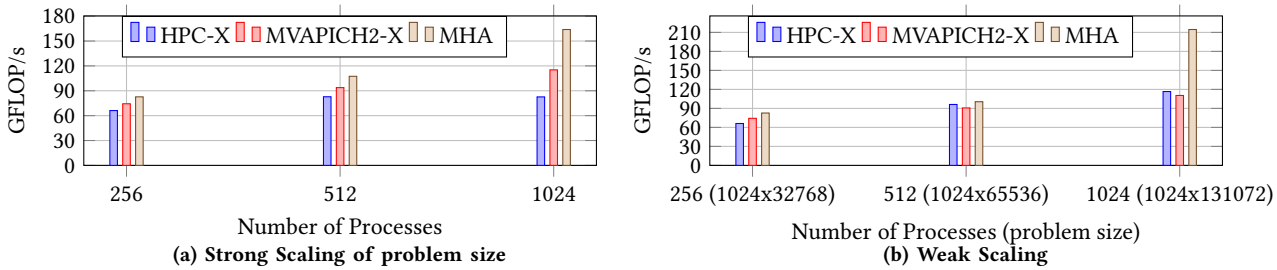
6 RELATED WORK

There are a few studies targeting optimization communication for multi-rail networks at collective level. Ying Qian et al. [28] proposed designs for RDMA-based Multi-port All-gather on multi-rail QsNet^{II} networks; our work here targets InfiniBand systems, but the designs are general and can be applied to any kind of network. Träff et al. [38] used a decomposition method to show that collectives can be redesigned for better performance when exploiting multi-lane



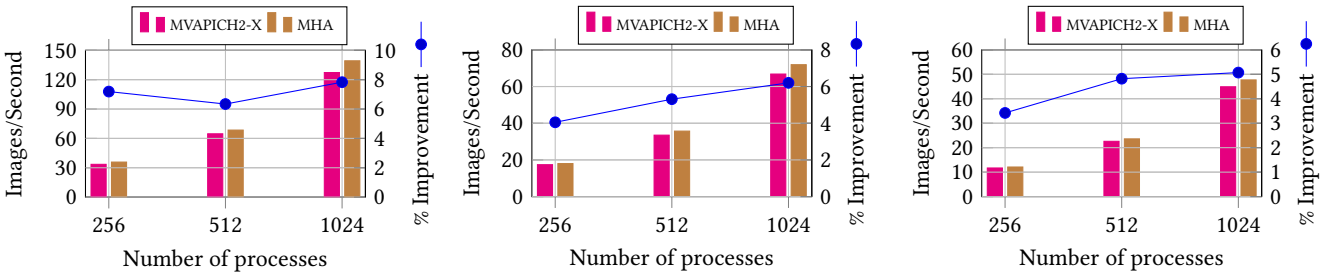
(a) 8 nodes (b) 16 nodes (c) 32 nodes

Figure 15: Evaluation of Proposed Inter-node MPI_Allreduce Design against state of the art libraries via OSU Microbenchmarks at scale (32 PPN)



(a) Strong Scaling of problem size 1024 x 32768 (b) Weak Scaling

Figure 16: Performance Evaluation of MHA against state of the art MPI libraries in a Matrix-Vector Multiplication kernel for Weak and Strong Scaling



(a) ResNet-50 (b) ResNet-101 (c) ResNet-152

Figure 17: Proposed MHA design against MVAPICH2-X via PyTorch + Horovod DL Performance Evaluation: Images Per Second

communication. Their work is considered to be a performance guideline to which users can refer when writing MPI programs running on multi-rail networks. Compared to this work, we propose designs that take low-level details into consideration and can be integrated into any of the existing MPI implementations. Users can directly invoke high-level functions like MPI_Allgather which take away the burden of performance from users.

There are several Allgather designs for single rail systems. Sur et al. [33] proposed an RDMA based All-to-all Broadcast (Allgather). Specifically, the design aims at eliminating the overhead of protocol handshake and multiple buffer registrations. Furthermore, they also cut down the copy cost by dynamically choosing an optimal threshold from a copy-based approach to a zero-copy one as the collective progresses. Mamidala et al. [19] proposed shared Memory and RDMA-based Design for Allgather. To clarify, communication buffers of each process using different communication channels are not shared; the authors use shared memory for sharing the buffers for both intra and inter-node communication, resulting in overlap of network operations with intra-node shared memory copies. Kandalla et al. [14] proposed multi-leader-Based Allgather algorithms for Multi-Core Clusters. Conventional flat and existing algorithms do not take into consideration of differences in latency and bandwidth of communication at inter-node, inter-socket, or intra-socket level, resulting in bottlenecks caused by the slowest communication level. The authors resolve the congestion by using multiple leaders per node to decouple communication at different levels.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose Multi-HCA aware designs for the Allgather collective operation. Furthermore, theoretical models to analytically study the impact of such designs are provided. For pure intra-node communication, by offloading some of the workload to the adapters, the performance improvement goes up to 65%. For inter-node communication with multiple processes per node, the

proposed hierarchical design with the use of shared memory for overlapping intra-node memory copies and network operations shows up to 71% improvement. In addition, Allreduce by utilizing the proposed Allgather delivers up to 44% reduction in latency. At the application level, a Matrix-Vector multiplication kernel using Allgather and a deep neural network application using Allreduce show 94% and 7.83% reductions in runtime, respectively. In the future, we plan to address other collectives and investigate the impact of NUMA systems on communication performance. The two-level design in this paper decouples intra-node and inter-node communication, but it is not NUMA-aware. We can have a 3-level design with the overlapping of intra-socket, inter-socket, and inter-node communication.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. {TensorFlow}: A System for {Large-Scale} Machine Learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*. 265–283.

[2] Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Sameer Kumar, Ewing Lusk, Rajeev Thakur, and Jesper Larsson Träff. 2009. MPI on a Million Processors. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 20–30.

[3] Mohammadreza Bayatpour, Jahanzeb Maqbool Hashmi, Sourav Chakraborty, Hari Subramoni, Pouya Kousha, and Dhabaleswar K Panda. 2018. Salar: Scalable and adaptive designs for large message reduction collectives. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 12–23.

[4] David E Bernholdt, Swen Boehm, George Bosilca, Manjunath Gorentla Venkata, Ryan E Grant, Thomas Naughton, Howard P Pritchard, Martin Schulz, and Geoffroy R Vallee. 2020. A survey of MPI usage in the US exascale computing project. *Concurrency and Computation: Practice and Experience* 32, 3 (2020), e4851.

[5] Adrián Castelló, Enrique S Quintana-Ortí, and José Duato. 2021. Accelerating distributed deep neural network training with pipelined MPI allreduce. *Cluster Computing* 24, 4 (2021), 3797–3813.

[6] Sourav Chakraborty, Hari Subramoni, and Dhabaleswar K Panda. 2017. Contention-aware kernel-assisted MPI collectives for multi-/many-core systems. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 13–24.

[7] Barbara Chapman, Tony Curtis, Swaroop Pophale, Stephen Poole, Jeff Kuehn, Chuck Koelbel, and Lauren Smith. 2010. Introducing OpenSHMEM: SHMEM for the PGAS community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*. 1–3.

[8] Daichi Mukunoki and Toshiyuki Imamura 2017. *Implementation and Evaluation of 2.5D Matrix Multiplication on the K computer*. Retrieved Mar 18, 2022 from https://prace-ri.eu/wp-content/uploads/PRACE-at-SC17-Daichi-Mokunoki.pdf

[9] Vijay Dhanraj. 2012. *Enhancement of LiMIC-Based Collectives for Multi-core Clusters*. Ph. D. Dissertation. The Ohio State University.

[10] El Capitan 2022. *El Capitan*. Retrieved Mar 18, 2022 from https://www.hpe.com/us/en/newsroom/press-release/2020/03/hpe-and-amd-power-complex-scientific-discovery-in-worlds-fastest-supercomputer-for-us-department-of-energys-doe-national-nuclear-security-administration-nnsa.html

[11] Frontier 2022. *Frontier*. Retrieved Mar 18, 2022 from https://www.olcf.ornl.gov/frontier/

[12] Jahanzeb Maqbool Hashmi, Sourav Chakraborty, Mohammadreza Bayatpour, Hari Subramoni, and Dhabaleswar K Panda. 2018. Designing efficient shared address space reduction collectives for multi-/many-cores. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1020–1029.

[13] HPC Advisory Council 2022. *Thor*. Retrieved Mar 18, 2022 from https://hpcadvisorycouncil.atlassian.net/wiki/spaces/HPCWORKS/pages/7864401/Thor

[14] Krishna Kandalla, Hari Subramoni, Gopal Santhanaraman, Matthew Koop, and Dhabaleswar K Panda. 2009. Designing multi-leader-based allgather algorithms for multi-core clusters. In *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 1–8.

[15] Keras 2022. *Keras Applications*. Retrieved Mar 18, 2022 from https://keras.io/api/applications/

[16] Ignacio Laguna, Ryan Marshall, Kathryn Mohror, Martin Ruefenacht, Anthony Skjellum, and Nawrin Sultana. 2019. A large-scale study of MPI usage in open-source HPC applications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–14.

[17] Jiuxing Liu, Abhinav Vishnu, and Dhabaleswar K Panda. 2004. Building multi-rail infiniband clusters: Mpi-level design and performance evaluation. In *SC'04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*. IEEE, 33–33.

[18] Teng Ma, George Bosilca, Aurelien Bouteiller, Brice Goglin, Jeffrey M Squyres, and Jack J Dongarra. 2011. Kernel assisted collective intra-node mpi communication among multi-core and many-core cpus. In *2011 International Conference on Parallel Processing*. IEEE, 532–541.

[19] Amith R Mamidala, Abhinav Vishnu, and Dhabaleswar K Panda. 2006. Efficient shared memory and RDMA based design for mpi_allgather over InfiniBand. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer, 66–75.

[20] Message Passing Interface Forum. 2021. *MPI: A Message-Passing Interface Standard Version 4.0*. https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[21] Paul Messina. 2017. The exascale computing project. *Computing in Science & Engineering* 19, 3 (2017), 63–67.

[22] OSU Micro-Benchmarks. 2018. Osu network-based computing laboratory. *URL: http://mvapich. cse. ohio-state. edu/benchmarks* 2 (2018).

[23] Network-Based Computing Laboratory 2022. *MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE*. Retrieved Mar 18, 2022 from http://mvapich.cse.ohio-state.edu/

[24] NVIDIA 2022. *HPC-X*. Retrieved Mar 18, 2022 from https://developer.nvidia.com/networking/hpc-x

[25] Open MPI 2022. *Open MPI: Open Source High Performance Computing*. Retrieved Mar 18, 2022 from https://www.open-mpi.org/

[26] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019).

[27] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J. Parallel and Distrib. Comput.* 69, 2 (2009), 117–124.

[28] Ying Qian and Ahmad Afsahi. 2007. High performance RDMA-based multi-port all-gather on multi-rail QsNet II. In *21st International Symposium on High Performance Computing Systems and Applications (HPCS'07)*. IEEE, 3–3.

[29] Ruslan Salakhutdinov and Andriy Mnih. 2008. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. In *Proceedings of the 25th international conference on Machine learning*. 880–887.

[30] Frank Seide and Amit Agarwal. 2016. CNTK: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2135–2135.

[31] Alexander Sergeev and Mike Del Balso. 2018. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799* (2018).

[32] Sameer S Shende and Allen D Malony. 2006. The TAU parallel performance system. *The International Journal of High Performance Computing Applications* 20, 2 (2006), 287–311.

[33] Sayantan Sur, Uday Kumar Reddy Bondhugula, Amith Mamidala, H-W Jin, and Dhabaleswar K Panda. 2005. High performance rdma based all-to-all broadcast for infiniband clusters. In *International Conference on High-Performance Computing*. Springer, 148–157.

[34] Rajeev Thakur, Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, Torsten Hoefler, Sameer Kumar, Ewing Lusk, and J Larsson Träff. 2010. MPI at Exascale. *Procceedings of SciDAC* 2 (2010), 14–35.

[35] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of collective communication operations in MPICH. *The International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[36] ThetaGPU 2022. *Theta/ThetaGPU Machine Overview*. Retrieved Mar 18, 2022 from https://www.alcf.anl.gov/support-center/theta/theta-thetagpu-overview

[37] Top500 2022. *NOVEMBER 2021*. Retrieved Mar 18, 2022 from https://www.top500.org/lists/top500/2021/11/

[38] Jesper Larsson Träff and Sascha Hunold. 2020. Decomposing MPI collectives for exploiting multi-lane communication. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 270–280.

[39] Tom Vander Aa, Imen Chakroun, and Tom Haber. 2017. Distributed Bayesian probabilistic matrix factorization. *Procedia Computer Science* 108 (2017), 1030–1039.

[40] Yili Zheng, Amir Kamil, Michael B Driscoll, Hongzhang Shan, and Katherine Yelick. 2014. UPC++: a PGAS extension for C++. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 1105–1114.

[41] Huan Zhou, José Gracia, and Ralf Schneider. 2019. MPI collectives for multi-core clusters: Optimized performance of the hybrid MPI+ MPI parallel codes. In *Proceedings of the 48th International Conference on Parallel Processing: Workshops*. 1–10.