Received: 13 December 2022 | Revised: 5 June 2023 | Accepted: 10 July 2023

DOI: 10.1002/cpe.7879

RESEARCH ARTICLE

WILEY

Accelerating communication with multi-HCA aware collectives in MPI

Tu Tran | **Bharath Ramesh** | **Benjamin Michalowicz** | **Mustafa Abduljabbar** | **Hari Subramoni** | **Aamir Shafi** | **Dhabaleswar K. Panda**

Computer Science and Engineering, The Ohio State University, Columbus, Ohio, USA

Correspondence
Tu Tran, Computer Science and Engineering, The Ohio State University, Columbus, OH, USA.
Email: tran.839@osu.edu

Summary

To accelerate the communication between nodes, supercomputers are now equipped with multiple network adapters per node, also referred to as HCAs (Host Channel Adapters), resulting in a "multi-rail"/"multi-HCA" network. For example, the ThetaGPU system at Argonne National Laboratory (ANL) has eight adapters per node; with this many networking resources available, utilizing all of them becomes non-trivial. 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 provide a thorough performance analysis of existing multirail solutions and their implications on collectives and present the necessity for further enhancement. Specifically, we propose novel designs for hierarchical, multi-HCA-aware Allgather. The proposed designs fully utilize all the available network adapters within a node and provide high overlap between inter-node and intra-node communication. At the micro-benchmark level, we see large inter-node improvements up to 62% and 61% better than HPC-X and MVAPICH2-X for 1024 processes. Because Allgather is used in Ring-Allreduce, our designs also improve its performance by 56% and 44% compared to HPC-X and MVAPICH2-X, respectively. At the application level, our enhanced Allgather shows 1.98× and 1.42× 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

Allgather, Allreduce, collectives, HCA-aware, MPI, network-aware

1 | INTRODUCTION

Modern supercomputers are now equipped with more than one network adapter per node, resulting in a multirail network to further increase bandwidth between compute nodes. Such a network can be built by using more than one Host Channel Adapter (HCA) per node. In this paper, the terms "rail" and "'HCA" are used interchangeably. Examples of such a system are Frontier[1] and El Capitan,[2] the first exascale computers planned to debut in 2022 and 2023. With enormous computing capabilities at our disposal, using *all* of these becomes a nontrivial task.

Among commonly used parallel programming models such as shared memory, message passing, and partitioned global address space (PGAS),[3,4] the Message Passing Interface (MPI) standard[5] is currently the de-facto parallel programming model on modern high-performance computing (HPC) clusters. The Exascale Computing Project (ECP)[6] was initiated as a primary attempt to pursue Exascale computing by the US government in 2016

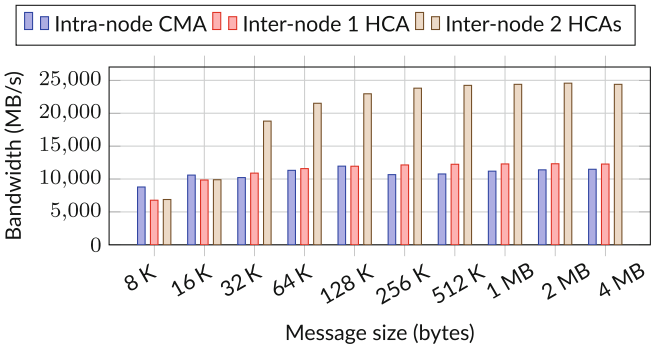

FIGURE 1 Bandwidth comparison between intra-node and inter-node communication.

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, etc. In 2018, Bernholdt et al.[7] 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.[8,9] Point-to-point and collective communications are the two main types defined by the MPI Standard; collectives, in particular, involve two or more processes and contribute to a significant part of many HPC applications' total runtime.

1.1 | Motivation

Inter-node communication is bolstered by increasing the number of network adapters per node; conversely, intra-node bandwidth will not change until/unless a cluster is upgraded to a 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 point-to-point communication using one and two HCAs at the MPI level. Here, the bandwidth of inter-node communication with one HCA is close 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.

In general, there are two categories of collective algorithms: flat and two-level. Conventional flat algorithms,[10] 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—namely, intra-node transfers. 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 and redrawn from the TAU profiling system.[11] TAU's tracing capability 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 continuously gets improved by the academic community[12-14]; it is frequently used in both traditional HPC and DL operations.

Two-level algorithms are also referred to as single-leader or multileader-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 broadcast the result to their intra-node peers. In the multi-leader-based design proposed in Reference 15, 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. 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.

1.2 | Contributions

In this paper, we take up these challenges and propose multi-HCA aware designs. In addition to preliminary results from a recently published study,[16] we do the following in addition to them: (1) We first study the performance characterization of existing multirail solutions. (2) With the insights,

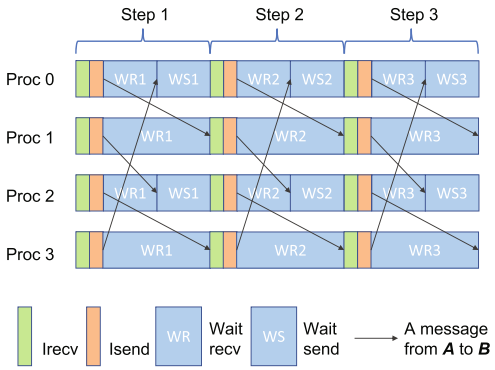

FIGURE 2 Allgather 2 Nodes, 2 PPN communication timeline visualization. Inter-node data exchanges between process 1, 2 and 0, 3 are two times faster than intra-node ones of process 0, 1 and 2, 3.

we then propose novel multirail-aware collectives. Specifically, we focus on the two significant ones: Allgather and Allreduce. Besides the wide application of Allreduce on both traditional HPC and Deep Learning, Allgather is an important collective used in many applications such as lower and upper triangle factorization, solving differential equations, linear algebra operations such as Bayesian Probabilistic Matrix Factorization,[17-19] and matrix–matrix/matrix–vector multiplication.[19,20] 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. To summarize, this paper makes the following contributions:

- We first evaluate the performance of existing multirail solutions to understand and identify their limitations.
- We propose scalable and hierarchical multi-HCA aware collectives.
- We conduct a thorough performance evaluation of the designs and compare them with various MPI implementations, namely HPC-X and MVAPICH2-X.
- Our new schemes achieve up to 62% improvement against HPC-X and 61% against MVAPICH2-X at the micro-benchmark level, and our design for inter-node communication boosts the performance of Ring Allreduce by 56% and 44% against HPC-X and MVAPICH2-X.
- At the application level, the enhanced Allgather shows 1.98× and 1.42× 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.

2 | PERFORMANCE ANALYSIS OF COLLECTIVES ON MULTIRAIL SYSTEMS

In this section, we first summarize the existing multi-rail designs: rail binding and rail sharing, then we evaluate and analyze the performance effect of them on collectives from both theoretical and experimental angles. From the results, we will demonstrate the need for multi-HCA aware designs for collectives.

2.1 | An overview of existing multirail designs

A point-to-point design at MPI level for multirail InfiniBand cluster was proposed in 2004 by Liu et al.[21] This is one of the first works to consider using multiple network adapters to speed up communication between nodes. Currently, most MPI implementations already have such support for multirail systems. To summarize, there are two basic adopted strategies: rail binding and rail sharing. Table 1 demonstrates the two mapping strategies of two adapters to four processes provided by MVAPICH2. (1) In rail binding, application developers can either have network adapters assigned to processes in a user-defined manner or in a round-robin fashion by default. (2) While, in rail sharing, as the naming suggests, each process has access to all the adapters residing within a node. 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 increases, the bandwidth usage of a rail will go up and eventually get saturated. The use of message 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. Figures 1 and 3 illustrate the bandwidth and latency tests between two processes on two nodes at the 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 a size greater than this. This cuts the latency of large messages in half.

TABLE 1 Network adapter-to-process binding under different strategies of four processes and two adapters per node.

Process	Rail binding	Rail sharing
0	mlx5_0	mlx5_0 mlx5_2
1	mlx5_2	mlx5_0 mlx5_2
2	mlx5_0	mlx5_0 mlx5_2
3	mlx5_2	mlx5_0 mlx5_2

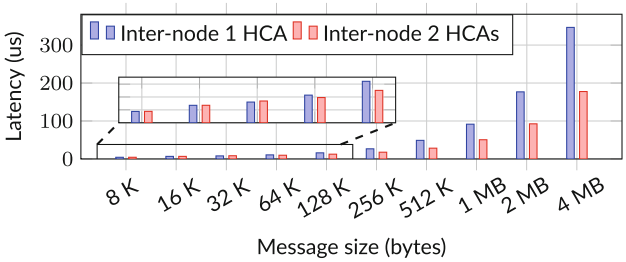

FIGURE 3 Inter-node latency comparison on two processes with one and two host channel adapters.

2.2 | Performance characterization of rail binding and rail sharing on collective algorithms

In the following subsections, detailed analyses of the performance of each collective are presented when running on a multirail cluster. To have better insights into why and when the existing multirail solutions work, performance models for collectives are also shown. In each collective, we focus on the analysis of the most commonly used algorithms that are designed to deliver good performance on different **targeted message ranges**: small, medium and large. There is no precise definition in separating message size in small, medium or large categories. In this paper, we consider **small** message size $<$ 1 KB, **medium** one $\geq$ 1 KB and $<$ 16 KB, and **large** one $\geq 16KB$. To demonstrate **the necessity for improvement of each algorithm**, we classify into three kinds: (1) **No** – existing solutions show benefits, (2) **Yes** – existing solutions show NO benefits even though an additional HCA is provided, but there is no slowdown, and (3) **Can be further improved** – existing solutions show benefits but it can be further improved to have better performance.

Table 2 summarizes the efficiency of different collective algorithms with rail binding and sharing of two adapters and compares them to the single-HCA case to quantify performance improvement and network resource utilization. The experimental numbers indicate that not every algorithm can be improved by rail binding or sharing. Three of the seven collectives need to be enhanced, and another one can potentially have better performance; those collectives and their corresponding message range are highlighted Table 2. Specifically, Ring Allgather and Allreduce, designed for large messages, show no benefit with either of the two existing multi-rail strategies, and all algorithms of Reduce listed here also have no improvement. In other words, in these cases, there is no benefit even though an additional network adapter is available. This indicates the network resources are not utilized efficiently, which presents a necessity for enhancement in such circumstances.

Table 3 depicts a list of notations that are used throughout this paper. To keep the analysis simple but straightforward and easy to understand, we employ Hockney model[23] to estimate the cost of collective algorithms, the total process count $N * P$ is a power of two, and omit delay caused by network congestion or memory contention, but retain the cost for adapter contention C_H between processes when its bandwidth is saturated. For rail binding of two HCAs, processes are mapped to HCAs in a round-robin fashion, each adapter serves $(P/2)$ processes, and C_H is reduced by half. For rail sharing, a message is broken into two chunks and sent over by two HCAs, so bandwidth BW_H is increased two times. A transfer of message M with H adapters can be modeled as $T_H(M) = \alpha_H + (C_H * M)/(BW_H * H)$. As we can see, with more adapters the latency term (α_H) is still the same, but the bandwidth term is divided by the number of adapters H. The model conforms well with empirical numbers of transfer messages using two adapters in Figure 3. In summary, when using more than one adapter to transfer messages, there will be an improvement for large messages, not for smaller ones.

2.2.1 | Allgather

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.

TABLE 2 Performance analysis of collective algorithms with existing multirail designs: rail binding and sharing.

Collective	Algorithm	Speedup of rail binding of two host channel adapters (HCAs)	Speedup of rail sharing of two HCAs	Targeted message range	In need of improvement
Allgather	Recursive doubling	1.95×	1.95×	Small message	No
	Direct Spread	2.24×	2×	Medium and large message	No
	Ring	**No improvement**	**No improvement**	**Large message**	**Yes**
Allreduce	Binomial tree	1.30×	1.30×	Small message	No
	Ring (Reduce Scatter – Allgather)	**No improvement**	**No improvement**	**Large message**	**Yes**
Scatter/ Gather	Binomial tree	No improvement	1.65×	Small message	No
	Direct	No improvement	1.95×	Large message	No
Bcast	Binomial tree	No improvement	1.23×	Small message	No
	Scatter – Recursive doubling Allgather	No improvement	1.25x	**Medium message**	**Can be further improved**
	Scatter – Ring Allgather	No improvement	1.14×	**Large message**	**Can be further improved**
Reduce	**Binomial tree**	**No improvement**	**No improvement**	**Small message**	**Yes**
	Reduce Scatter – Gather	**No improvement**	**No improvement**	**Large message**	**Yes**
Alltoall	Bruck	1.30×	1.37×	Small message	No
	Pairwise	1.71×	1.92×	Large message	No

Notes: The collectives showing no improvement and needed to enhance are highlighted in bold. The experiments are conducted on Thor cluster (Section 5.1), running with 256 processes with the message size from 1 B to 1 MB and using MVAPICH2-X software.

TABLE 3 Notations used in the cost models.

Symbol	Description
N	Number of **N**odes
P	Number of **P**rocesses per node
M	**M**essage size
H	Number of adapters (**H**CAs)
γ	Computation cost per byte for reduction
$\alpha_{\mathbf{C}}$	Startup time per intra-node transfer by **C**PU
$\mathrm{BW}_{\mathbf{C}}$	Bandwidth of intra-node transfer by **C**PU
$\alpha_{\mathbf{H}}$	Startup time per inter-node transfer by **H**CA
$\mathrm{BW}_{\mathbf{H}}$	Bandwidth of inter-node transfer by **H**CA
$\alpha_{\mathbf{L}}$	Startup cost per **L**ocal memory copy
$\mathrm{BW}_{\mathbf{L}}$	Bandwidth of **L**ocal memoy copy
C_H	Cost of network adapter contention
$T_C(M)$	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

1. Recursive Doubling (RD) executes in $\log_2(P * N)$ steps with $\log_2(P)$ intra-node transfers and $\log_2(N)$ inter-node transfers for each process, where N and P are the number of nodes and local 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. RD is good for small messages, and its total cost is $T_{\text{RD}}(M) = \log_2(P)\alpha_C + \log_2(N)\alpha_H + \frac{C_H * M}{BW_C}\sum_{i=1}^{\log_2 P} 2^{i-1} + \frac{C_H * M}{BW_H}\sum_{j=\log_2 P+1}^{\log_2 P * N} 2^{i-1}$. Figure 4A shows that for both rail binding and sharing, as the message size goes up, we see 2× better than using one rail. For rail binding, each adapter serves only $(P/2)$ processes, which leads to the speedup. On the other hand, with rail sharing, every process has access to two adapters, so the transfer time of a message M is 2× faster. As a result, the overall time of RD is 1.95× faster.
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 $N * P$ processes participating in the communication, the algorithm will take $N * P - 1$ steps to finish. For this particular algorithm, the slowest links are the bottlenecks because other processes cannot continue until they have received the data traveling through these slow links. Therefore, the total cost is $T_{\text{Ring}}(M) = (N * P - 1) * Max(\alpha_C + \frac{C_H * M}{BW_C}, \alpha_H + \frac{C_H * M}{BW_H})$. With rail binding and rail sharing, only inter-node bandwidth is increased, so neither of it can help improve communication for this algorithm.
3. In Direct Spread (dissemination) Allgather algorithm, each process directly communicates with the source processes to get the data instead of getting from an intermediate process the way Ring algorithm does. Specifically, in step i, a process rank r receives data directly from the process rank $(r - i)\%N$. This algorithm requires $N * P - 1$ steps to complete like Ring for a communicator of $N * P$ processes. The total cost of Direct Spread is $T_{DS}(M) = (P - 1)(\alpha_C + \frac{C_H * M}{BW_C}) + (N - 1)P(\alpha_H + \frac{C_H * M}{BW_H})$. Figure 4B demonstrate the speedups are 2.24× and 2× for rail binding and sharing, respectively. With 2× inter-node bandwidth due to an additional adapter, rail binding is able to provide better than 2× performance due to the fact that each adapter only handles $(P/2)$ processes, which leads to a 2× reduction in the number of send and receive requests each adapter has to handle. Long queue length leads to long expected waiting time in the queue. On the other hand, the number of send and receive requests for rail sharing and the single-HCA cases are the same.

Figure 5 compares the performance of Allgather algorithms when combined with the multirail strategy that shows the best benefits. All of them are configured to use two HCAs but Ring because it shows no benefit. For messages >2KB, Ring delivers the best performance with just one rail. As discussed earlier, with additional adapters, the latency term (α) remains, but the bandwidth is divided by the number of adapters partaking in the communication. As a result, it is very challenging to improve small message range, but it is not the case for medium and large ones. As a result, this presents an opportunity for optimization.

2.2.2 | Allreduce

The two most commonly used Allreduce algorithms are Binomial tree for small messages and Ring for large ones.

1. Similar to Recursive Doubling, the Binomial Tree pattern executes in $\log_2(P * N)$ steps with $\log_2(P)$ intra-node data exchanges and $\log_2(N)$ inter-node exchanges. The differences are data are reduced at each step and data size is not doubled. As a result, the communication cost is $T_{\text{Bin}}(M) = \log_2(P)(\alpha_C + \frac{C_H * M}{BW_C} + M\gamma) + \log_2(N)(\alpha_H + \frac{C_H * M}{BW_H} + M\gamma)$. The empirical results in Figure 6 indicate that the improvement of existing designs

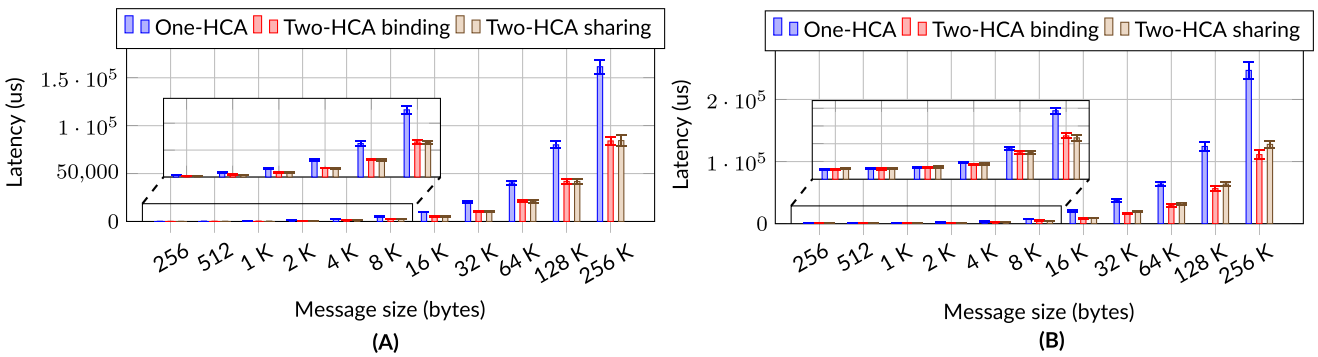

FIGURE 4 Performance comparison of Allgather algorithms with one and two host channel adapters (256 B to 256 KB). While there is no performance difference for the three strategies when message size <256B, multirail strategies continue to show benefits for message size >256 KB. (A) Recursive doubling; (B) direct spread.

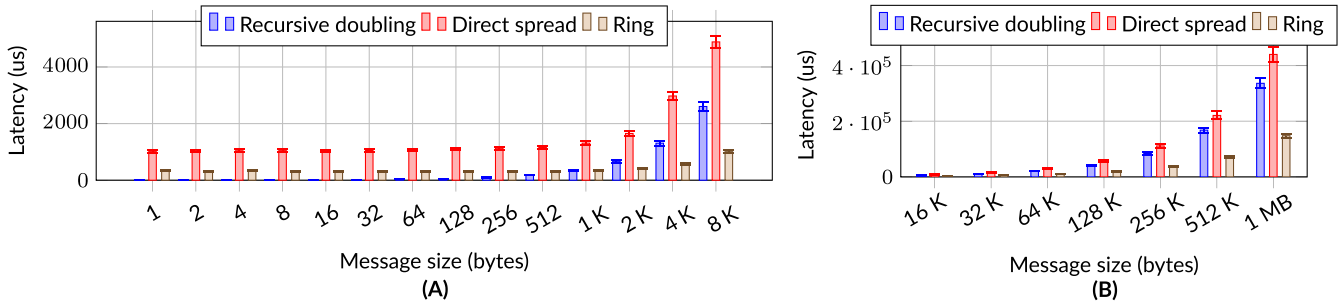

FIGURE 5 Comparison of Allgather algorithms when using the best multirail strategy (two host channel adapters) for each (1 B to 1 MB). (A) Small and medium messages; (B) large messages.

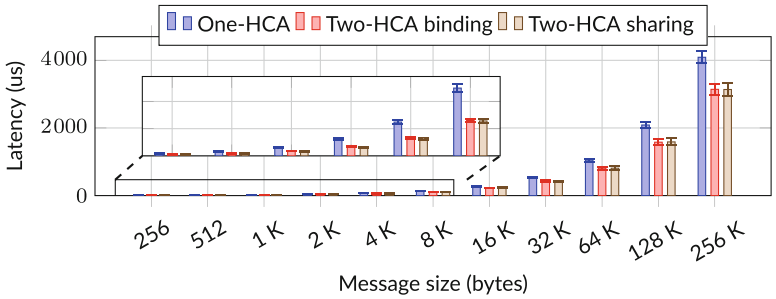

FIGURE 6 Performance of Binomial tree with one and two host channel adapters (Allreduce) (256 B to 256 KB). While there is no performance difference for the three strategies when message size <256 B, multirail strategies continue to show benefits for message size >256 KB.

is up to 1.30× better than 1-HCA case, owing to the facts that each adapter handles 2× less load for rail binding case and the time to deliver a message is 2× faster for rail sharing.

2. Proposed by Patarasuk et al.,[24] Ring Allreduce is proven to be bandwidth-optimal, making it 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, a logical ring communication is performed in $N * P - 1$ iterations to exchange data of size $(\frac{M}{N*P})$. As analyzed previously, in the ring pattern, the slowest links dictate the whole communication. Therefore, the cost of Reduce-Scatter phase is $T_{R-S} = (N * P - 1) * Max(\alpha_C + \frac{C_H * M}{N*P*BW_C} + \frac{M}{N*P}\gamma, \alpha_H + \frac{C_H * M}{N*P*BW_H} + \frac{M}{N*P}\gamma)$. After that, a Ring Allgather is performed, and as a result, each process will have a complete reduction vector. The total cost of Ring Allreduce is a sum of T_{R-S} and $T_{RingAllgather}$, discussed earlier. The experimental results in Table 2 show rail binding and sharing do not help improve the performance. This situation is similar to the case of Ring Allgather, and there is potential for improvement.

2.2.3 | Scatter/gather

Scatter is the reverse of Gather and vice versa. They both use the same algorithms for small and large messages, (binomial tree and direct send/receive), respectively.

1. The binomial tree pattern of a Scatter operation is similar to the one of Allreduce, which happens in $\log_2(P * N)$ steps with $\log_2(N)$ inter-node transfers and $\log_2(P)$ intra-node data transfers. In each step i, the root process and the processes that received data in previous steps send data to processes that are $(\frac{N*P}{2^i})$ ranks away. In contrary to Binomial tree Allreduce, there is no reduction, and the transfer data size is divided by half in each step. A Gather is a reverse of this algorithm. The overall cost is $T_{Bin}(M) = \log_2(N)\alpha_H + \sum_{i=1}^{\log_2 N} \frac{C_H * M * N * P}{BW_H 2^i} + \log_2(P)\alpha_C + \sum_{j=\log_2 N*P+1}^{\log_2 N*P} \frac{C_H * M * N * P}{BW_C 2^j}$. Figure 7A shows that as message size is large enough (>2 kB), the speedup is up to 1.65× for rail sharing and none for the binding case. By the nature of this algorithm, only one process in a node does the inter-node transfers, so allowing it to have access to one HCA in rail binding shows no improvement over one-HCA case
2. For large messages, the root process posts $(N * P - 1)$ nonblocking sends to transfer data of size M directly to other processes. Since intra-node transfers are performed by CPU cores, and inter-node ones are performed by adapters, they can be overlapped. As a result, the communication cost is $T_{Direct}(M) = Max\left[(P-1)(\alpha_C + \frac{C_H * M}{BW_C}), (N-1)(\alpha_H + \frac{C_H * M}{BW_H})\right]$. Figure 7B indicates that the speedup is up to 1.95× for messages from 32 kB to 1 MB rail sharing and none for the binding case. For this collective, only the root has data, so rail binding does not help with performance.

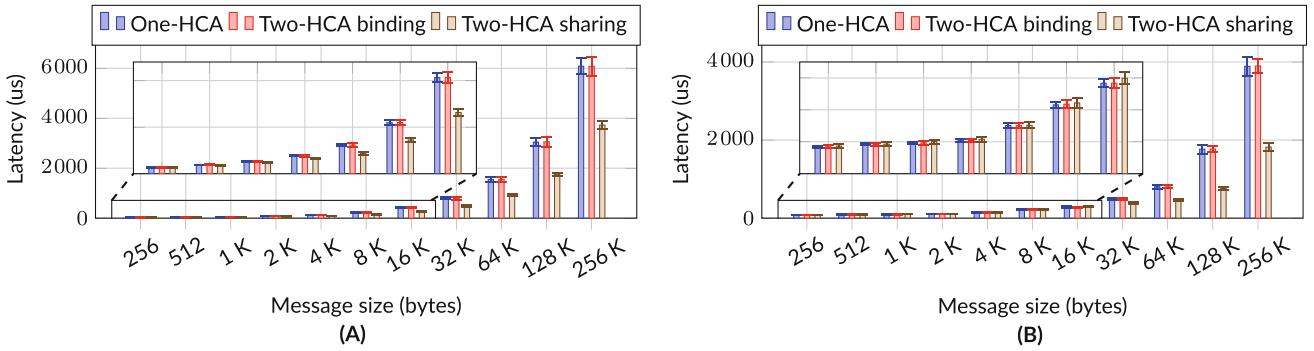

FIGURE 7 Performance comparison of scatter algorithms with one and two host channel adapters (256 B to 256 kB). While there is no performance difference for the three strategies when message size <256 B, multirail strategies continue to show benefits for message size >256 kB. (A) Binomial tree; (B) direct.

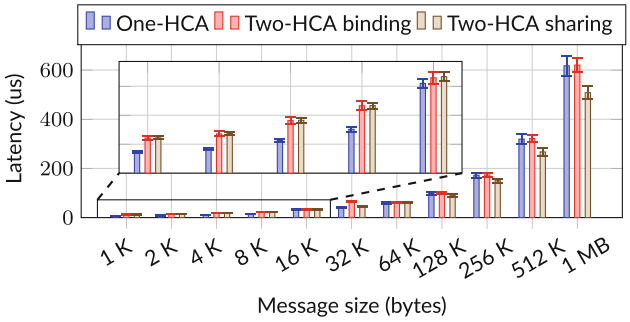

FIGURE 8 Performance of Binomial tree Bcast with one and two host channel adapters (HCAs) (1 KB to 1 MB). For message size from 1 B to 32 kB, multi-rail strategies with design overhead lead to slightly higher latency than the single HCA case.

2.2.4 | Bcast

Similar to other collectives, Bcast also utilizes binomial tree communication pattern for small messages. For larger ones, a Scatter is first performed, and then an Allgather is followed.

1. The way Bcast communicates using binomial tree pattern is the same as Scatter. At each step i, processes with data send to processes that are $(\frac{N*P}{2^i})$ ranks away. The only distinction is that the data size remains the same for all steps. Therefore, the total cost is $T_{\text{Bin}}(M) = \log_2(N)(\alpha_H + \frac{C_H*M}{\text{BW}_H}) + \log_2(P)(\alpha_C + \frac{C_H*M}{\text{BW}_C})$. The speedup of rail sharing is up to 1.23× for messages from 256 kB in Figure 8. Similar to the case of Scatter, rail binding is not beneficial because only one process does inter-node transfers.
2. For larger message size, Bcast is a combination a Scatter followed by a recursive doubling or ring Allgather, since the cost of each component has already been analyzed previously, the total cost is straightforward $T_{\text{Bcast}}(M) = T_{\text{Scatter}}(M) + T_{\text{Allgather}}(M)$. Even though Figure 9 shows that rail sharing improves the performance, a speedup of 1.14–1.25× for messages from 128 kB, Bcast can be further improved by optimizing Allgather, which is the focus of this paper. In addition, large-message Bcast algorithms show no improvement with rail binding.

2.2.5 | Reduce

A variant of Allreduce, Reduce also uses binomial tree for a small message range. For a large one, the approach is also similar to ring Allreduce. Specifically, a Reduce first performs a Reduce-Scatter, and then is followed by a Gather.

1. Since binomial trees of Reduce and Allreduce are the same. the cost is also the same $T_{\text{Bin}}(M) = \log_2(P)(\alpha_C + \frac{C_H*M}{\text{BW}_C} + M\gamma) + \log_2(N)(\alpha_H + \frac{C_H*M}{\text{BW}_H} + M\gamma)$. Since only one process does the inter-node transfer, rail binding is not useful. For rail sharing, the empirical numbers show no improvement. Besides communication time, computation time also takes up a portion of the total time for this collective.

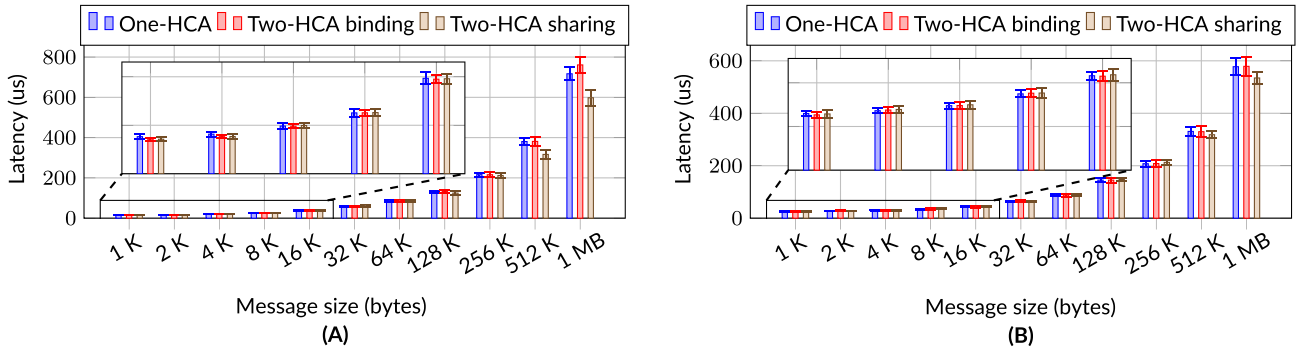

FIGURE 9 Performance comparison of Bcast algorithms with one and two host channel adapters (1 kB to 1 MB). For message size <1K, there is no performance difference for the three strategies. (A) Scatter – recursive doubling Allgather; (B) Scatter – Ring Allgather.

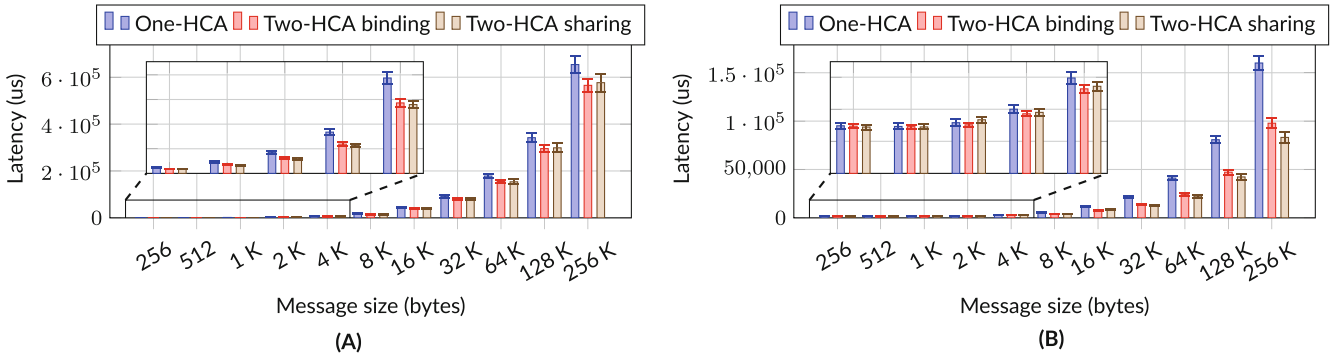

FIGURE 10 Performance of Alltoall algorithms with one and two host channel adapters (256 B to 256 kB). While there is no performance difference for the three strategies when message size <256 B, multirail strategies continue to show benefits for message size >256 kB. (A) Bruck; (B) pairwise.

2. The cost for large-message Reduce algorithm, a Reduce-Scatter followed by a Gather, is just the sum of each collective. Therefore, to obtain the final cost of Reduce, we can just replace the cost of Allgather with Gather (Section 2.2.3) in the final cost of Ring Allreduce in Section 2.2.2. The empirical numbers show a marginal improvement of just 1.03×. The small improvement here comes from the Gather phase. Thus, we consider there is no improvement when using existing multirail strategies.

2.2.6 | Alltoall

There are not many algorithms for Alltoall such as Bruck, Scatter Destination, and Pairwise. Here we focus on Bruck for small messages and Pairwise for large ones.

1. Bruck is a store-and-forward algorithm, good for small messages. Before performing any data transfer, all processes do a local data shift. After that, the algorithm executes in $\log_2(N * P)$ steps. In step i ($0 \leq i < \log_2(N * P)$), process rank r sends data of size $M\frac{N*P}{2}$ to rank $r + 2^i$ and receives from rank $r - 2^i$. When the communication is done, another local inverse rotation is performed to place data in order.[10] In the final cost model, we include the time to do inverse and rotation (T_{localop}) for an Alltoall of $N * P$ processes with message M, which becomes significant when M is large. The total cost of the algorithm is $T_{\text{Bruck}}(M) = \log_2(P)(\alpha_C + \frac{C_H * M * N * P}{2\text{BW}_C}) + \log_2(N)(\alpha_H + \frac{C_H * M * N * P}{2\text{BW}_H}) + T_{\text{localop}}(M, N * P)$. Figure 10A indicates a speedup of up to 1.30× and 1.37× for rail binding and sharing with message size from 512 B to 1 MB.
2. Pairwise algorithm executes in $(N * P - 1)$ steps with $(P - 1)$ and $(N - 1) * P$ exchanges with local and remote processes, respectively. In each step i, process rank r exchanges data of size M with rank ($rxori$). As a result, the total cost is $T_{\text{Pairwise}}(M) = (P - 1)(\alpha_C + \frac{C_H * M}{\text{BW}_C}) + (N - 1)P(\alpha_H + \frac{C_H * M}{\text{BW}_H})$. A speedup of up to 1.71× and 1.92× for rail binding and sharing with message size from 2 kB to 1 MB is shown in Figure 10B.

In summary, not every collective can be improved by existing multi-rail solutions (rail binding or sharing), such as Allgather, Allreduce, Reduce. Since Allgather is a common factor in other collectives, enhancing Allgather will lead to the improvement of other collectives: Allreduce and Bcast.

In this paper, owing to the popularity of Allgather and Allreduce, we focus on optimizing Allgather and showcasing its benefit on Allreduce. More precisely, novel designs are proposed to make Allgather multi-HCA aware.

3 | THE PROPOSED DESIGNS

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

In Allgather, each process sends its data to every other process. For pure intra-node communications, while these send operations are performed by memory copy operations that are executed by CPU processors, network adapters remain completely idle. As a result, each process can further accelerate communication by assigning a fraction of their workload to these adapters. Figure 11 demonstrates the design idea in which intra-node communication is accelerated by H HCAs. Since the number of processes is usually larger than the HCA count per node, the offloaded work coming from processes should be evenly distributed among HCAs so that each process finishes all of its send requests at the same time.

By assigning too much or too little work to processors or adapters, communication will be hampered by the component taking the longest time to finish its task. Therefore, in order for each process and adapter to finish roughly at the same time, we must ensure that they handle the appropriate workload. Here a tuning algorithm is proposed to track down an ideal workload to offload to adapters. Figure 12 shows the relationship between the offload size to adapters and the time it takes to finish the communication. We can easily determine the optimal point by first measuring the duration of communication done by all adapters while processors remain idle. After that, the offloaded workload is gradually reduced until the intersection of the communication latency's downward and upward trend lines is reached based on the correlation.

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:

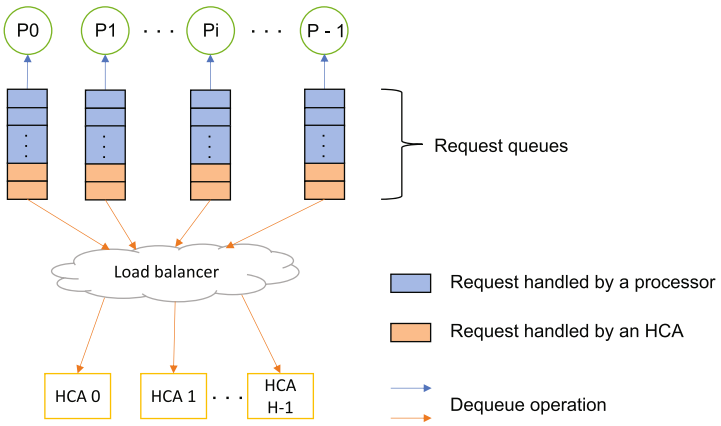

FIGURE 11 Multi-HCA aware design: Utilizing idle host channel adapters (HCAs) to accelerate intra-node communication.

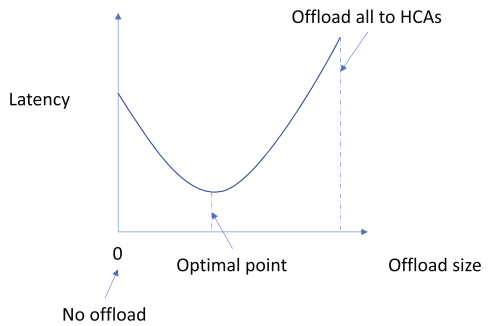

FIGURE 12 Correlation between the offload size to adapters and latency.

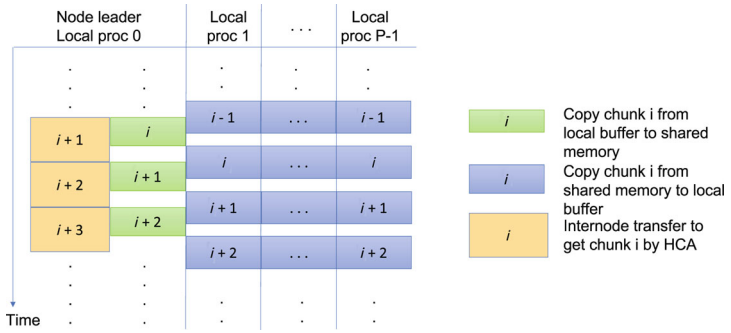

FIGURE 13 A timeline view of communication events of a node during inter-leader data exchange and node-level data distribution phases.

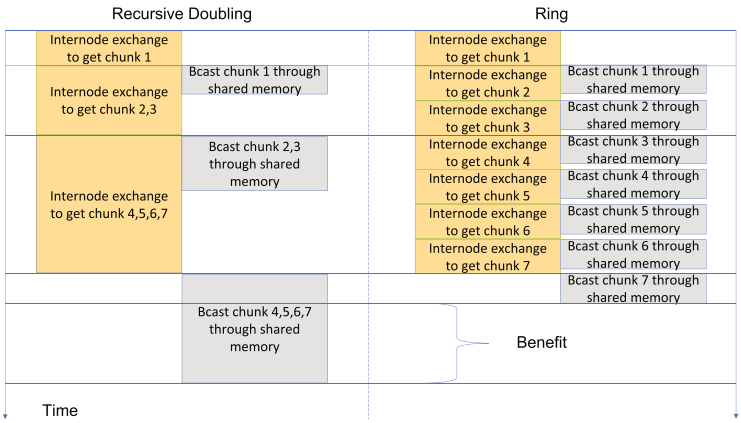

FIGURE 14 A timeline view of communication events of a node during inter-leader data exchange and node-level data distribution phases.

- 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.

During intra-node communication in phase one, network adapters are usually idle due to memory copies being used in lieu of the network, which leads to inefficient resource utilization. The proposed intra-node Allgather is used to have better utilization of network resources. In phase two, inter-leader data exchanges of N nodes can be done in $\log N$ steps with RD or in $(N-1)$ steps with Ring. Node-level data distribution in phase three can be overlapped with phase two by using shared memory. As soon as a data chunk arrives in each step in phase two, 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. Thus, network transfers and intra-node memory copies can be overlapped, which is demonstrated in Figure 13.

For inter-leader data exchanges in phase two, Ring can perform better and deliver more overlap than RD, depending on message size. Figure 14 depicts the case of 8 leaders corresponding to eight nodes with Ring outperforming RD due to higher overlap. For RD, The size of data transferred in the current step is twice that of the previous step, hence 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 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 increases, we see a better level of overlap delivered by Ring when compared to RD. Figure 15 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. 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.

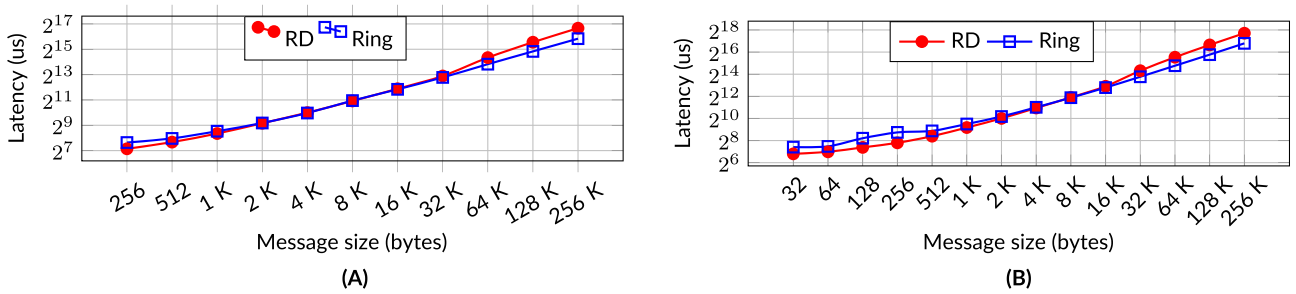

FIGURE 15 Comparison of the recursive doubling and Ring algorithms in the proposed design during inter-leader data exchange. (A) 16 Nodes, 32 PPN; (B) 32 nodes, 32 PPN.

4 | PERFORMANCE MODELS OF THE MULTI-HCA AWARE DESIGNS

4.1 | Modeling the cost of MHA-intra Allgather

Suppose there are P processes participating in an Allgather operation. In the MHA-intra algorithm, each process first copies its 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 $(P-1-d)$ intra-node transfers. d is the optimal number of offloaded transfers to HCAs per process, depending on the number of processes P 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:

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

As mentioned previously, a transfer of message M by H adapters can be modeled as $T_H(M) = \alpha_H + M/(\mathrm{BW}_H * H)$. A local memory copy of size M can be modeled as $T_P(M) = \alpha_P + M/\mathrm{BW}_P$. An intra-node transfer of message M can be modeled as $T_C(M) = \alpha_C + (M/\mathrm{BW}_C) * b$, 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:

$$\begin{aligned} T_{\text{MHA-intra}}(M) = T_P(M) &\\ + Max\{&(P-1-d) * T_C(M), \\ &P * d * T_H(M)\}. \end{aligned} \tag{2}$$

4.2 | Modeling the cost of MHA-inter Allgather

For MHA-inter Allgather, the communication happens in three phases. In phase 1, data are shared with group leaders using MHA-intra algorithm, then the cost is $T_{\text{MHA-intra}}(M)$, modeled in the previous section. For phase 2, group leaders perform data exchange of size $(M * P)$, either using RD or Ring. While RD runs with $\log N$ steps with data size doubled in every step, Ring executes in $N-1$ steps with data size of $(M * P)$. Then, the cost for phase two is

$$\begin{aligned} T_{\text{phase2-RD}}(MP) &= T_{\text{step1}} + T_{\text{step2}} + \cdots + T_{\text{step log}(N)} \\ &= T_H(M * P) + T_H(2 * M * P) \\ &\quad + \cdots + T_H(\log(N) * M * P) \\ &= \alpha_H * \log(N) \\ &\quad + (N-1) * (M * P)/(\mathrm{BW}_H * H). \end{aligned} \tag{3}$$

$$\begin{aligned} T_{\text{phase-2-Ring}}(MP) &= T_{\text{step1}} + T_{\text{step2}} + \cdots + T_{\text{step}(N-1)} \\ &= T_H(M * P) + T_H(M * P) \\ &\quad + \cdots + T_H(M * P) \\ &= \alpha_H * (N-1) \\ &\quad + (N-1) * (M * P)/(\text{BW}_H * H). \end{aligned} \tag{4}$$

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

$$\begin{aligned} &T_{\text{intra bcast}}(M * P) \\ &\quad = T_{\text{copy in}}(M * P) + T_{\text{copy out}}(M * P) \\ &\quad = (\alpha_P + (M * P)/\text{BW}_P) \\ &\qquad + (\alpha_P + (M * P)/\text{BW}_P) * cg(M * P, P-1). \end{aligned} \tag{5}$$

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

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

.

$$\begin{aligned} &T_{\text{MHA-inter-Ring}}(M) \\ &\quad = T_{\text{phase1}} + T_{\text{phase2}} + T_{\text{intra bcast}}(M * P), \\ &\qquad \text{if } T_{\text{intra bcast}}(M * P) \leq T_H(M * P) \\ &\quad = T_H(M * P) + (N-1) * T_{\text{intra bcast}}(M * P), \\ &\qquad \text{otherwise} \end{aligned} \tag{7}$$

4.3 | Model validation

To predict the performance of MHA-intra and MHA-inter Allgather, we must first empirically obtain parameters in Table 3. For intra-node communication, Equation (2) is used to estimate the cost of MHA-intra. Figure 16 shows that the predicted latency is close to the actual latency of MHA-intra, which means the proposed model can estimate the trend properly. Since MHA-intra is designed for large messages, and the larger the message is the more benefit it delivers, so a large message range from 256 K to 16 MB is presented in Figure 16.

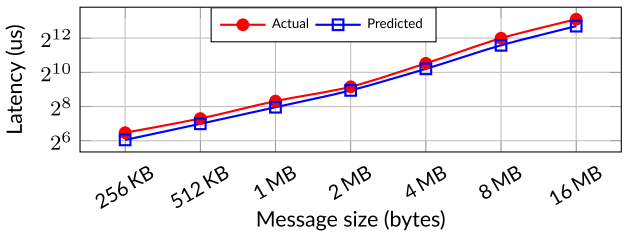

FIGURE 16 Validation of MHA-intra with four processes.

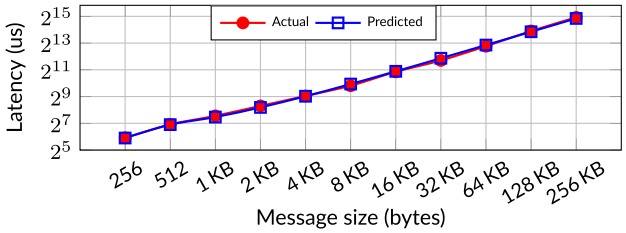

FIGURE 17 Validation of MHA-inter with eight nodes 32 PPN.

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 17, 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 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 *P* PPN on a system of *H* adapters.

5 | PERFORMANCE EVALUATION

We first present the environment for evaluation and then provide the results of experiments performed to evaluate our proposed designs at the micro-benchmark and application levels.

5.1 | Experimental environment

We conducted our experiments on the "Thorε partition of HPC Advisory Council's cluster.[25] 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 100 Gb/s InfiniBand adapters and 256 GB DDR4 2400 MHz 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.

The proposed designs are compared with two widely-used MPI libraries: MVAPICH2-X-2.3[26] and HPC-X-2.10.0.[27] 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 10/11, and Rockport Networks networking technologies. NVIDIA® HPC-X® is a variant of OpenMPI[28] maintained by NVIDIA that provides high performance, scalability, and efficiency and ensures that communication is fully optimized for NVIDIA InfiniBand networking solutions. At micro-benchmark level, we use OSU Micro-Benchmarks (OMB),[26] which is widely adopted by both academic and industrial communities for benchmarking MPI performance. For evaluating DL performance, we use PyTorch-1.8.0[29] and Horovod-0.20.0.[30] For a higher statistical confidence, all of the experiments are run five times, and any noise or fluctuation has already been filtered out. Within each OMB run, each message is an average of 1000 iterations for message size <8192 B and 100 iterations for larger ones. Due to the nature of different designs, while some targets large and very large messages, others deliver good performance for medium and large messages, thus different message ranges are shown: (1) a message range from 256 kB to 16 MB for for intra-node Allgather in Section 5.2, (2) a message range from 256 B to 256 kB for inter-node Allgather in Section 5.3, and (3) a message range from 128 kB to 128 MB for Allreduce in Section 5.4.

5.2 | Intra-node Allgather evaluation

Figure 18A–D shows the performance evaluation of Allgather with different numbers of processes using OSU micro-benchmarks.[22] The proposed design with the assistance of two 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. We note an expected trend: as the number of processes in the communication increases given a constant number of adapters, the performance benefit decreases. Each process offloads a portion of its workload to HCAs with the objective that processes and HCAs can finish at the same time, to reduce communication time. The offloaded portion gets smaller with more participating processes because the HCAs also must 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. Hence more adapters are needed for sustained performance when more processes are involved in the communication.

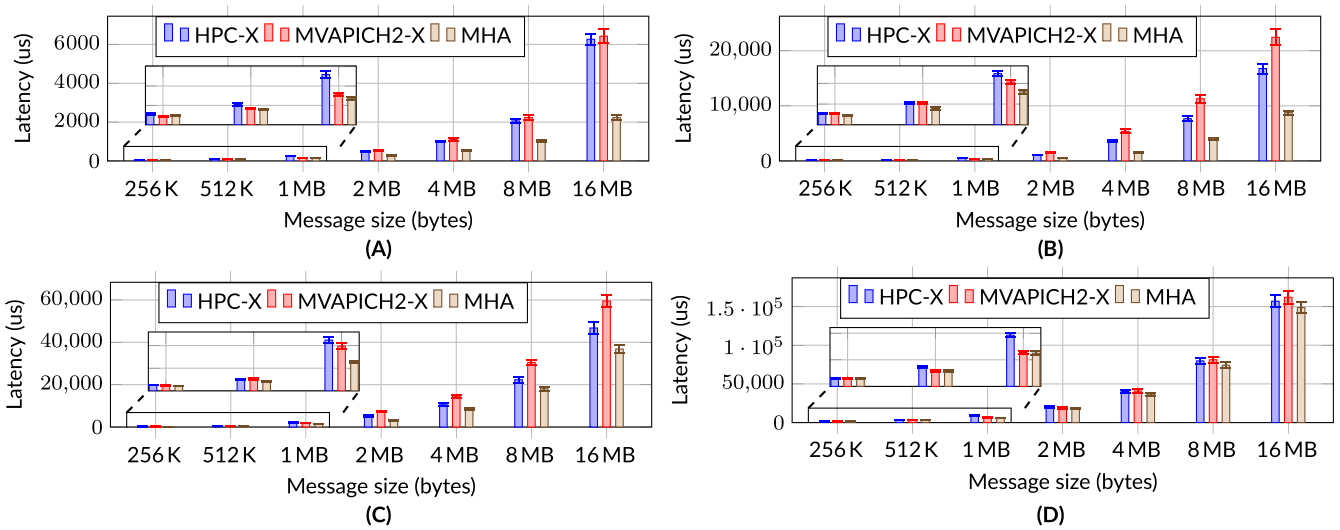

FIGURE 18 Evaluation of proposed Intra-node MPI_Allgather design against state of the art libraries via OSU Microbenchmarks. (A) 2 Processes; (B) 4 processes; (C) 8 processes; (D) 16 processes.

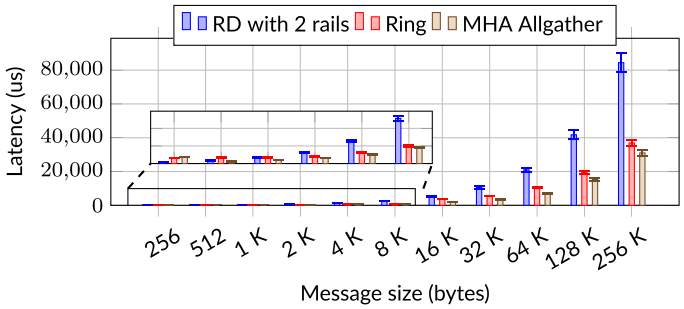

FIGURE 19 Performance of different Allgather algorithms with the proposed design on 256 processes.

5.3 | Inter-node Allgather evaluation

We perform inter-node experiments with 8, 16, and 32 nodes, up to full-subscription of each node (32 PPN). The proposed design is initially compared against RD (with two HCAs) and Ring. Figures 19 and 20 show that MHA Allgather is 47% better than RD at small messages and 41% better than Ring at large messages—where each algorithm excels, respectively. Figures 21,22, and 23 compare the performance of our designs with MVAPICH2-X and HPC-X when running with 256, 512, and 1024 processes, respectively. Our designs show significant improvement against MVAPICH2-X and HPC-X: 29% and 21% better for 256 processes, 44% and 53% better for 512 processes, and 62% and 61% better for 1024 processes, respectively.

By decoupling inter-node and intra-node communication with a single leader per node in the proposed design, multiple HCAs are efficiently utilized for communication across nodes. Additionally, performance gains also come from a higher overlap provided by Ring during the inter-node distribution phase. The numbers shown are tuned numbers between these two algorithms.

5.4 | Accelerating Allreduce with MHA Allgather

Allgather is used by several collectives, among them being Allreduce: in a Ring-Allreduce, a reduce-scatter is first performed, followed by an Allgather, and through improving Allgather, we also improve Allreduce. Figure 24A,B shows an average of 10% and 26% improvement of the enhanced Ring-Allreduce at 256 and 512 processes. As the message size goes up to 128 MB, the improvement linearly decreases because of (1) the increase in reduction time of larger buffers and (2) the higher cost of additional memory copies to shared regions when data no longer fit into cache.

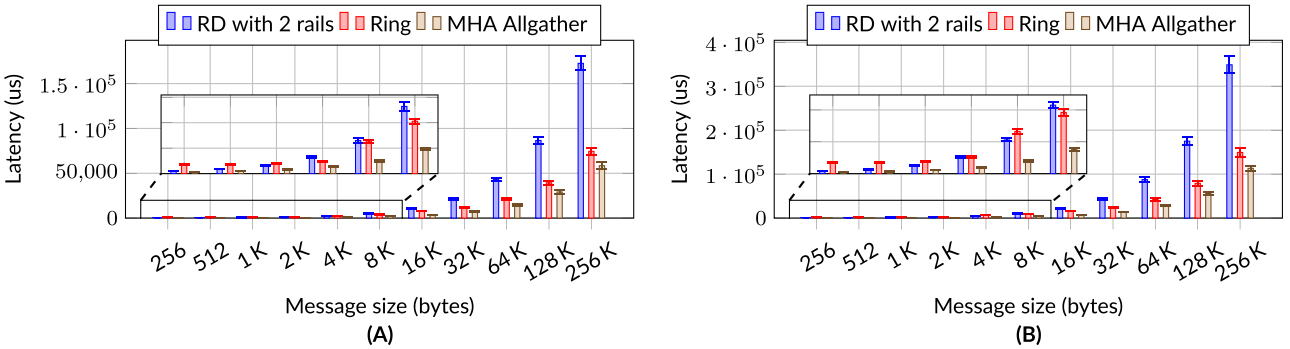

FIGURE 20 Performance of different Allgather algorithms with the proposed design. (A) 512 processes; (B) 1024 processes.

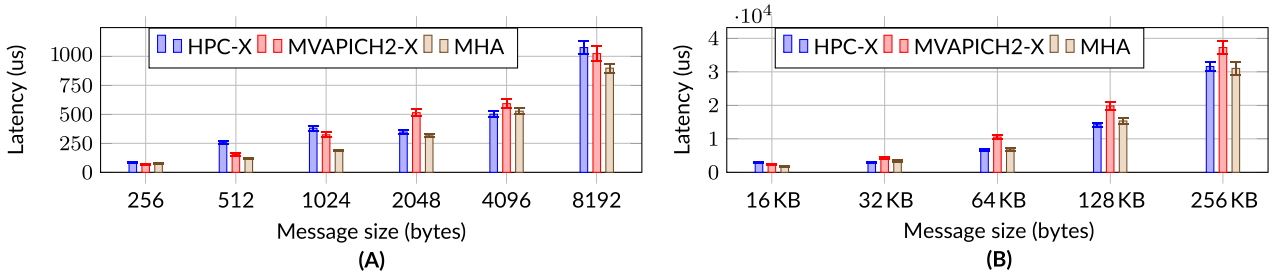

FIGURE 21 Proposed MPI_Allgather against state of the art libraries via OSU Microbenchmarks on 256 processes (eight nodes 32 PPN). (A) Medium messages; (B) large messages.

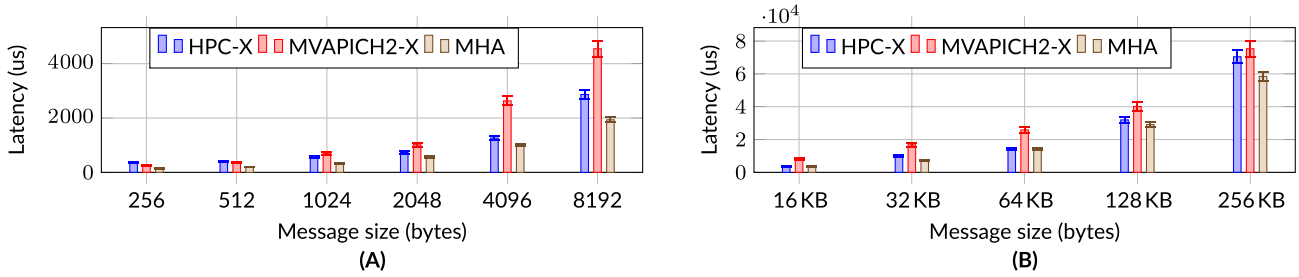

FIGURE 22 Proposed MPI_Allgather against state of the art libraries via OSU Microbenchmarks on 512 processes (16 nodes 32 PPN). (A) Medium messages; (B) large messages.

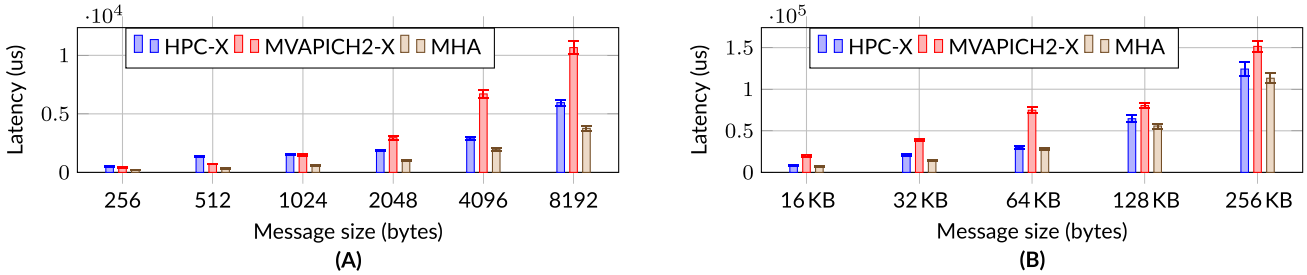

FIGURE 23 Proposed MPI_Allgather against state-of-the-art libraries via OSU Microbenchmarks on 1024 processes (32 nodes 32 PPN). (A) Medium messages; (B) large messages.

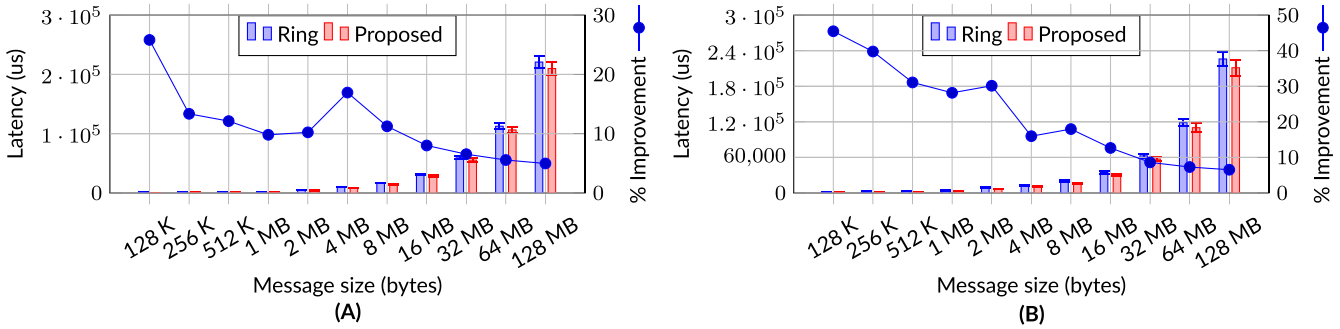

FIGURE 24 Performance comparison of the proposed design with Ring Allreduce. (A) 256 processes; (B) 512 processes.

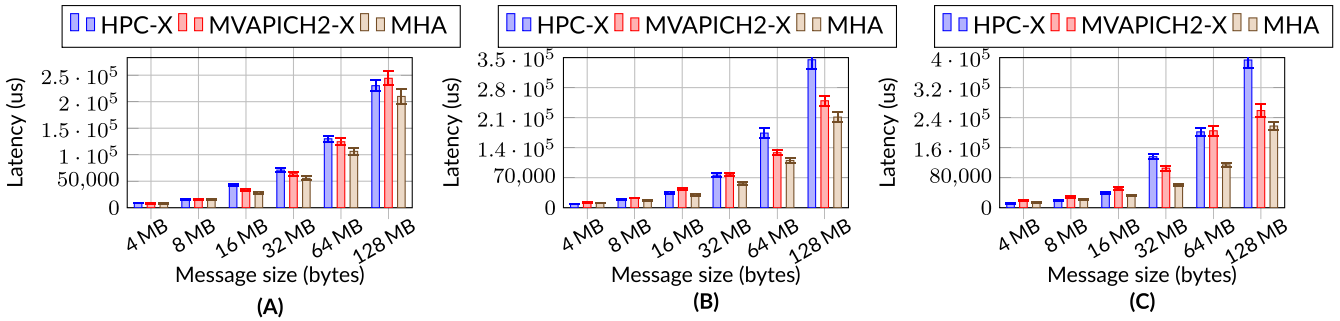

FIGURE 25 Evaluation of proposed inter-node MPI_Allreduce design against state of the art libraries via OSU Microbenchmarks at scale (32 PPN). (A) 8 nodes; (B) 16 nodes; (C) 32 nodes.

Figure 25 depicts the performance of the improved Allreduce compared to HPC-X and MVAPICH2-X. We see that Allreduce performs up to 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.

5.5 | Impact of MHA Allgather on matrix-vector multiplication

Allgather appears in many applications: lower/upper triangle factorization, differential equations, linear algebra operations such as Bayesian Probabilistic Matrix Factorization,[17,18] and matrix-matrix/matrix-vector multiplication.[19,20] 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 $M * N$, X and Y are inputs, and the output vector is of size $N * 1$ and $M * 1$, respectively. A is partitioned using one-dimensional row layout, in which each process holds ($M/\#of$processes) rows. Similarly, vector x any y are broken into equal segments of size ($N/\#of$processes) and ($M/\#of$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 26 demonstrates the performance of the matrix-vector multiplication kernel in GFLOP/s (higher is better). In these experiments, we configure the problem size ($M * N$) so that communication contributes a significant time in the total runtime of the kernel to see the impact of the improved Allgather. The proposed Allgather outperforms both HPC-X and MVAPICH2-X by up to 1.98× and 1.42× for strong scaling and 1.84× and 1.94× for weak scaling experiments with 1024 processes.

5.6 | Impact of the improved Allreduce on deep learning training

Here, we compare the runtime of training different neural networks through PyTorch and Horovod. In particular, 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 million parameters, respectively.[31] Due to technical issues, we cannot set up HPC-X to work with Pytorch + Horovod despite our best effort. The reason may be that Open MPI cannot work with several versions of Horovod, reported on Horovod's website. Figure 27 shows that as the number of processes increases, we observe up to a 7.83% improved runtime than MVAPICH2-X in both epoch time and images per second for ResNet50. We see similar benefits when switching to a larger neural network (ResNet101 or ResNet152).

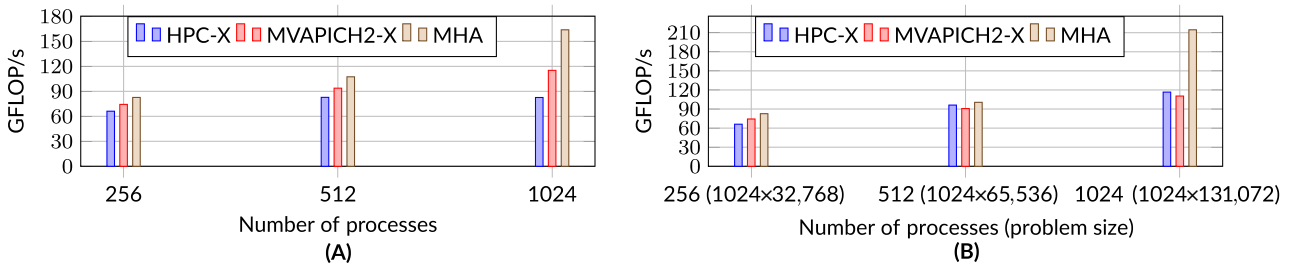

FIGURE 26 Performance evaluation of MHA against state of the art MPI libraries in a matrix-vector multiplication kernel for weak and strong scaling. (a) Strong Scaling of problem size 1024 × 32,768; (B) weak scaling.

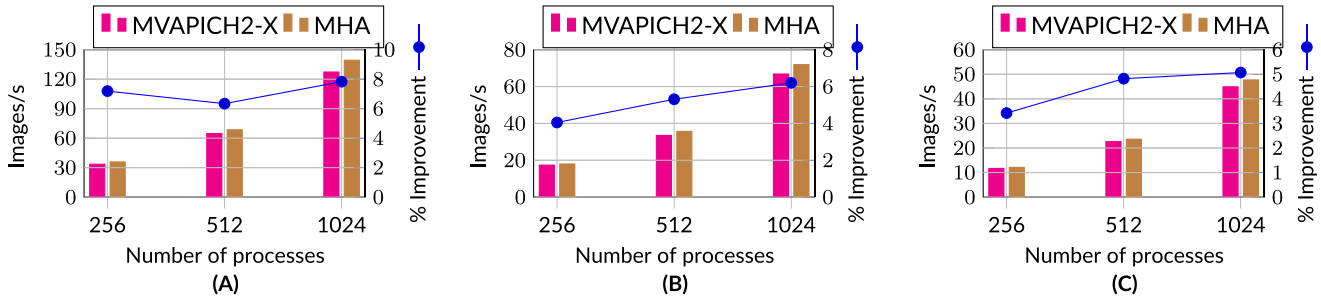

FIGURE 27 Proposed MHA design against MVAPICH2-X via PyTorch + Horovod DL performance evaluation: images per second. (A) ResNet-50; (B) ResNet-101; (C) ResNet-152.

6 | RELATED WORK

There are a few studies targeting optimization communication for multirail networks at collective level. The work of Chan et al.[32] is similar to ours, in which they proposed new collective algorithms where a node can send data over multiple links. However, their work, dated back to 2006, targeted direct networks (router-based) while ours is on indirect networks (switched). Nowadays, router-based networks are very rare, and most systems are built based on switch-based networks. Qian et al.[33] proposed designs for RDMA-based Multi-port All-gather on multirail QsNet^{II} networks; ours target InfiniBand systems, but the designs are general and can be applied to any kind of network. Träff et al.[34] used a decomposition method to show that collectives can be redesigned for better performance when exploiting multilane communication. Their work is considered to be a performance guideline users can reference when writing MPI programs for on multirail networks. By comparison, we propose designs that take low-level details into consideration and can be integrated into any existing MPI implementation. 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.[35] proposed an RDMA-based All-to-all Broadcast (Allgather). Specifically, the design aims at eliminating the overhead of protocol handshakes 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.[36] proposed shared Memory and RDMA-based Design for Allgather. The 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 overlapping of network operations with intra-node shared memory copies. Kandalla et al.[15] proposed multi-leader-based Allgather algorithms for multicore clusters. Conventional flat and existing algorithms do not take into consideration of differences in latency and bandwidth of communication at the 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 thoroughly analyze the performance of existing multirail solutions on collectives, theoretically and empirically. Out of the seven mentioned, three require improvement, and another one can be further enhanced. Based on the insights and analyses, we propose multi-HCA aware designs for the Allgather collective operation with performance models to analytically study the impact of such designs. We also show how Allgather can be utilized to improve other collectives such as Allreduce. By offloading some of the workload to the adapters in intra-node communication, we

see up to 65% performance gains. Our proposed hierarchical designs, which overlap with shared memory/intra-node communication, exhibit up to 71% performance gains. In addition, Allreduce delivers up to 44% reduction in latency by utilizing our proposed Allgather. At the application level, the Matrix-Vector multiplication kernel using Allgather and a DL application using Allreduce show 94% and 7.83% reductions in runtime, respectively. In the future, we plan to address other collectives such as Reduce and investigate the impact of NUMA systems on communication performance.

ACKNOWLEDGMENTS

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

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Tu Tran https://orcid.org/0000-0003-0040-8404

REFERENCES

1. Frontier. Accessed March 18. 2022 https://www.olcf.ornl.gov/frontier/
2. El Capitan. Accessed March 18. 2022. 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
3. Zheng Y, Kamil A, Driscoll MB, Shan H, Yelick K. UPC++: a PGAS extension for C++. IEEE 28th International Parallel and Distributed Processing Symposium; 2014:1105-1114.
4. Chapman B, Curtis T, Pophale S, et al. Introducing OpenSHMEM: SHMEM for the PGAS community. Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model; ACM; 2010:1-3.
5. Message Passing Interface Forum. MPI: a message-passing interface standard version 4.0. 2021.
6. Messina P. The exascale computing project. *Comput Sci Eng*. 2017;19(3):63-67.
7. Bernholdt DE, Boehm S, Bosilca G, et al. A survey of MPI usage in the US exascale computing project. *Concurr Comput Pract Exp*. 2020;32(3):e4851.
8. Balaji P, Buntinas D, Goodell D, et al. MPI on a million processors. *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer; 2009:20-30.
9. Thakur R, Balaji P, Buntinas D, et al. MPI at exascale. *Proc SciDAC*. 2010;2:14-35.
10. Thakur R, Rabenseifner R, Gropp W. Optimization of collective communication operations in MPICH. *Int J High Perform Comput Appl*. 2005;19(1):49-66.
11. Shende SS, Malony AD. The TAU parallel performance system. *Int J High Perform Comput Appl*. 2006;20(2):287-311.
12. Castelló A, Quintana-Orti ES, Duato J. Accelerating distributed deep neural network training with pipelined MPI allreduce. *Cluster Comput*. 2021;24(4):3797-3813.
13. Bayatpour M, Hashmi JM, Chakraborty S, Subramoni H, Kousha P, Panda DK. Salar: scalable and adaptive designs for large message reduction collectives. IEEE International Conference on Cluster Computing (CLUSTER); 2018:12-23.
14. Hashmi JM, Chakraborty S, Bayatpour M, Subramoni H, Panda DK. Designing efficient shared address space reduction collectives for multi-/many-cores. IEEE International Parallel and Distributed Processing Symposium (IPDPS); 2018:1020-1029.
15. Kandalla K, Subramoni H, Santhanaraman G, Koop M, Panda DK. Designing multi-leader-based allgather algorithms for multi-core clusters. IEEE International Symposium on Parallel & Distributed Processing; 2009:1-8.
16. Tran T, Michalowicz B, Ramesh B, Subramoni H, Shafi A, Panda DK. Designing hierarchical multi-HCA aware Allgather in MPI. Workshop Proceedings of the 51st International Conference on Parallel Processing; ACM; 2022:1-10.
17. Salakhutdinov R, Mnih A. Bayesian probabilistic matrix factorization using Markov chain Monte Carlo. Proceedings of the 25th International Conference on Machine Learning; ACM; 2008:880-887.
18. Vander Aa T, Chakroun I, Haber T. Distributed Bayesian probabilistic matrix factorization. *Proc Comput Sci*. 2017;108:1030-1039.
19. Zhou H, Gracia J, Schneider R. MPI collectives for multi-core clusters: optimized performance of the hybrid MPI+ MPI parallel codes. Proceedings of the 48th International Conference on Parallel Processing: Workshops; ACM; 2019:1-10.
20. Implementation and evaluation of 2.5D matrix multiplication on the K computer. 2017. Accessed March 18, 2022 https://prace-ri.eu/wp-content/uploads/PRACE-at-SC17-Daichi-Mokunoki.pdf
21. Liu J, Vishnu A, Panda DK. Building multirail infiniband clusters: Mpi-level design and performance evaluation. SC'04: Proceedings of the 2004 ACM/IEEE Conference on Supercomputing; 2004:33-33.
22. OSU Micro-Benchmarks. Osu network-based computing laboratory. Accessed March 18, 2022. http://mvapich.cse.ohio-state.edu/benchmarks
23. Hockney RW. The communication challenge for MPP: intel paragon and Meiko CS-2. *Parallel Comput*. 1994;20(3):389-398.
24. Patarasuk P, Yuan X. Bandwidth optimal all-reduce algorithms for clusters of workstations. *J Parallel Distrib Comput*. 2009;69(2):117-124.
25. Thor. Accessed March 18. 2022 https://hpcadvisorycouncil.atlassian.net/wiki/spaces/HPCWORKS/pages/7864401/Thor
26. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. Accessed March 18. 2022 http://mvapich.cse.ohio-state.edu/
27. HPC-X. Accessed March 18. 2022 https://developer.nvidia.com/networking/hpc-x
28. Open MPI: Open source high performance computing. Accessed March 18. 2022 https://www.open-mpi.org/
29. Paszke A, Gross S, Massa F, et al. Pytorch: an imperative style, high-performance deep learning library. *Adv Neural Inform Process Syst*. 2019;8024-8035.
30. Sergeev A, Del Balso M. Horovod: fast and easy distributed deep learning in TensorFlow. *arXiv preprint arXiv:1802.05799*. 2018.
31. Keras Applications. Accessed March 18. 2022 https://keras.io/api/applications/
32. Chan E, Van De Geijn R, Gropp W, Thakur R. Collective communication on architectures that support simultaneous communication over multiple links. Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming; 2006:2-11.

33. Qian Y, Afsahi A. High performance RDMA-based multi-port all-gather on multi-rail QsNet II. 21st International Symposium on High Performance Computing Systems and Applications (HPCS'07); 2007:3.
34. Träff JL, Hunold S. Decomposing MPI collectives for exploiting multi-lane communication. -2020 IEEE International Conference on Cluster Computing (CLUSTER); 2020:270-280.
35. Sur S, Bondhugula UKR, Mamidala A, Jin HW, Panda DK. High performance rdma based all-to-all broadcast for infiniband clusters. *International Conference on High-Performance Computing*. Springer; 2005:148-157.
36. Mamidala AR, Vishnu A, Panda DK. Efficient shared memory and RDMA based design for mpi_allgather over InfiniBand. *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting*. Springer.; 2006:66-75.

How to cite this article: Tran T, Ramesh B, Michalowicz B, et al. Accelerating communication with multi-HCA aware collectives in MPI. *Concurrency Computat Pract Exper*. 2023;e7879. doi: 10.1002/cpe.7879