OHIO: Improving RDMA Network Scalability in MPI_Alltoall through Optimized Hierarchical and Intra/Inter-Node Communication Overlap Design

Tu Tran, Goutham Kalikrishna Reddy Kuncham, Bharath Ramesh, Shulei Xu,
Hari Subramoni, Mustafa Abduljabbar, Dhabaleswar K. (DK) Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, USA
{tran.839, kuncham.2, ramesh.113, xu.2452, subramoni.1, abduljabbar.1}@osu.edu,
panda@cse.ohio-state.edu

***Abstract*—The presence of exascale computers has pushed a new boundary in computing capability, which poses performance challenges in parallel programming models on how to exploit such systems efficiently. A dominant programming model for running parallel programs is the Message Passing Interface. Among primitives provided by MPI, Alltoall is a communication-intensive operation, which is utilized by many applications and is well-known for being difficult to optimize. Alltoall algorithms can be mainly classified into flat and hierarchical. The hierarchical designs avoid the slowdown of intra-node communication by inter-node communication by decoupling them. The hierarchical designs also reduce network congestion by reducing concurrently injected messages into the network. This work demonstrates an additional benefit of hierarchical designs to improve connection scalability in RDMA networks. This is attributed to the cache thrashing happening inside network adapters. All of these advantages of hierarchical schemes collectively contribute to the network scalability of Alltoall. This motivates us to propose a further optimized hierarchical design to enhance performance and network scalability. The design is network-agnostic and evaluated on clusters with InfiniBand and Omni-Path network adapters. The proposed design achieves average latency improvements of 61.13%, 56.40%, 37.49%, and 51.90% over Open MPI + UCX, HPC-X, Intel MPI, and MVAPICH2-X at micro-benchmark level with up to 7168 cores, respectively. In addition, the evaluation at application-level with Car-Parrinello Molecular Dynamics code shows 24.98%, 40.44% and 50.48% improvement in the simulation time, compared to MVAPICH2-X, Open MPI + UCX, and Intel MPI, respectively.**

***Index Terms*—Alltoall, Collective, MPI, Optimization, Performance, Scalability, Network Agnostic, Transport Protocol**

I. Introduction

Contemporary high-performance clusters are equipped with powerful CPUs with a high processor count per node. An AMD system featuring Milan or Rome architectures supports up to 128 cores per node, while the Intel Ice Lake architecture offers a maximum of 80 cores. Such nodes are then connected together with high-performance networks such as InfiniBand, Omni-Path, and RoCE. These networks distinguish themselves from traditional Ethernet with Remote Direct Memory Access (RDMA) in providing high-performance zero-copy and kernel bypass message transfers. To fully exploit supercomputers, an efficient and robust programming model is required to catch up with the trend of such systems in the number of core counts per node, network speed, and system size.

Among parallel programming models, Message Passing Interface (MPI) [1] is the de-facto model that is utilized on many high-performance clusters to execute applications in a parallel and distributed fashion. The MPI standard is continuously evaluated by its community to closely follow the trend of modern clusters. MPI provides a set of primitives for point-to-point and collective communication. Among collective operations, Alltoall is the most communication-intensive collective that injects a huge amount of data into the network; it is not only utilized in traditional high-performance computing applications [2], [3] but also in emerging Artificial Intelligence/ Machine Learning applications [4], [5]. Alltoall is well-known for being difficult to scale and optimize due to its global communication pattern. As a result, providing good Alltoall performance and scaling it on a large RDMA network is not trivial, especially at the exascale level with millions of cores.

RDMA networks play an important role in optimizing communication, especially Alltoall. Many studies have shown that RDMA networks suffer performance degradation as the number of connections crosses a certain threshold [6]–[9]. The root cause of the issue in RDMA network is attributed to the cache thrashing inside the adapters. An RDMA network adapter, also known as RDMA NIC (RNIC) or Host Channel Adapter (HCA), can create a certain number of connections, represented by QPs (Queue Pairs), and cache a subset of them within its local cache. Figure 1 illustrates RNIC architecture and how it uses its SRAM to cache information, such as virtual-to-physical address translation for the registered memory regions as well as QP-related information. To retrieve the connection-related information, the adapter has to read from the CPU's main memory, leading to an increase in traffic going through PCIe. As reported in [9], InfiniBand ConnectX-4 and ConnectX-5 have a cache size of around $2MB$. In Mellanox's implementation, each RC connection consumes $375B$, allowing the RNIC to accommodate approximately 5000 connections ($1.8MB$). In addition, there is an upper bound on the number of QPs an RDMA adapter can create. For instance, the max number of QPs supported on InfiniBand ConnectX-6 adapters is 131072. However, this number is considered small, considering the exascale era we have entered. Several studies have shown that performance degradation happens even sooner, after a certain threshold, i.e. 256 QPs, which is significantly less than the estimated number (5000 QPs) [7], [8].

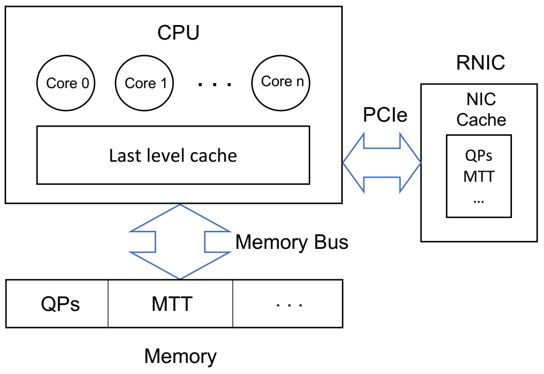


Fig. 1: RDMA NIC (RNIC) Architecture. RNIC caches QP-related information, as well as the memory translation table (MTT). Upon NIC cache misses, RNIC fetches the data from main memory via DMA over PCIe.

The connection scalability issue can be overcome with other types of transport such as unreliable datagrams (UD). Specifically, while the throughput of RC drops significantly as the number of connections increases, the throughput of UD remains constant [10], [11]. This motivates the adoption of UD in communication runtime like MPI. However, UD neither guarantees packet transmission nor provides packet reordering. This requires software solutions that add some extra overhead and can lead to performance degradation. Besides RC and UD, Dynamic Connection (DC) is another transport that attempts to retain both the performance of RC and the scalability of UD [12]. However, this transport protocol is specific to InfiniBand and not supported in Omni-Path, RoCE, or other networks.

A. *Motivation*

An Alltoall operation of an N-node network requires $O(N^2)$ number of messages and its algorithms can mainly classified into flat and hierarchical schemes. For flat algorithms such as Bruck [13], they work well for single-processor nodes; communication is performed at the network level [14]. Over time, more resources are put into a processor; hence the arrival of multi/many-core processors. The number of messages in an Alltoall increases up to $O(N^2 * PPN^2)$ with PPN as the number of cores per node. This leads to the creation of hierarchical designs attributed to the performance difference with processor interconnect (intra-node) faster than network interconnect (inter-node) [15], [16]. This is the first benefit of hierarchical designs - avoiding communication slowdown when decoupling intra-node and inter-node communication. Another benefit is the reduction in network congestion with less concurrently injected messages into the network [16]–[18]. In hierarchical schemes, there will be a leader within a node in charge of gathering data and communicating on behalf of all processes in the node. The number of messages sent to the network decreases from $O(N^2 * PPN^2)$ to $O(N^2)$.

In fact, hierarchical designs provide an additional benefit related to the connection scalability in RDMA networks besides the two well-studied advantages commonly cited in the literature. They together contribute to the network scalability of Alltoall. RDMA network adapters face a scalability issue as the number of reliable connections (RC) scales up [7], [8]. In order to send a message to a remote process, an RC connection must be established. Hierarchical designs decrease the number of connections from $(N-1) * PPN$ to $(N-1)$ when compared to flat designs. As a result, this leads to a reduction in the number of PCIe read transactions to retrieve connection data structures stored in main memory. This motivates us to further improve hierarchical designs for overall network scalability due to the three aforementioned benefits. In this work, we explore, analyze the benefit from RDMA connection scalability angle that is not well-known in MPI literature, and propose an optimized hierarchical design.

Table I summarizes well-known alltoall algorithms and highlights their differences with the final proposed design. While some algorithms are hierarchical, other increases resource utilization by overlapping intra-node and inter-node communication. Nevertheless, few prior works take into account of both connection scalability in an RDMA network and communication overlap when designing Alltoall algorithms, which leads to low performance at a large scale. More importantly, there is a plethora of works that study the scalability of such networks and the interplay between network transport protocols and communication performance, notably [6], [7], [11], [20]. Our proposed design overcomes the challenge by reducing the number of QPs as well as posted requests per node while preserving the benefits of hierarchy and communication overlap.

B. *Contribution*

To the best of our knowledge, this is the first work that studies the benefit of hierarchical designs from connection scalability angle in the context of MPI for Alltoall. **In this work, we propose OHIO - an Optimized Hierarchical and Intra/Inter-Node Communication Overlap design to improve the network scalability of Alltoall with a focus on small messages. The proposed solution applies to any RDMA network. It not only scales as well as UD but also maintains all the reliability of RC.** We thoroughly discuss and analyze the techniques used in the designs. They are evaluated with micro-benchmark and application. As a result, this paper makes the following contributions:

1) We propose high-performance and scalable network-aware Alltoall design.
2) In addition, the design itself is network-agnostic. In this paper, we have taken InfiniBand and Omni-Path networks as a case study.

TABLE I: Existing and Proposed Designs for Alltoall Communication

Algorithm	Optimal message range	Hierarchical	Intra/inter-node communication Overlap	Number of QPs used per node	Number of posted send requests per node
Bruck [13]	Small	✗	✗	$\sim PPN * log_2(N * PPN)$	$\sim PPN * log_2(N * PPN)$
Scatter Destination	Large	✗	✗	$(N-1) * PPN^2$	$(N-1) * PPN^2$
Pairwise	Large	✗	✗	$(N-1) * PPN^2$	$(N-1) * PPN^2$
Khorassani et al. [19]	Large	✓	✓	$(N-1) * PPN$	$(N-1) * PPN$
Träff et al. [15]	Small	✓	✗	$(N-1)$	$(N-1)$
Chochia et al. [16]	Small	✓	✗	$(N-1)$	$(N-1)$
Proposed Design	Small	✓	✓	$(N-1)$	$(N-1)$

TABLE II: Characteristics of Reliable Connection (RC) and Unreliable Diagram (UD) in InfiniBand Adapters

Transport Protocol	MTU Size	One-sided Verbs			Two-sided Verbs
		Read	Write	Atomic	Send/Recv
RC	2 GB	✓	✓	✓	✓
UD	4 KB	✗	✗	✗	✓

3) We conduct a thorough performance evaluation of the design with various MPI implementations, namely Intel MPI, Open MPI, HPCX, and MVAPICH2 on clusters with different types of interconnects using micro-benchmarks.
4) The scalability of the design is also studied against different transport protocols, namely, RC, DC, and UD.
5) The design shows benefits at application-level with Car-Parrinello Molecular Dynamics code when compared against other MPI implementations.

II. Background

A. Remote Direct Memory Access (RDMA)

RDMA is a network feature that bypasses the kernel to allow direct access to the memory of a remote host, enabling zero-copy transfers with low latency. It offers high bandwidth and reduces CPU cycles for network operations. Popular high-performance RDMA networks include InfiniBand (IB), RDMA over Converged Ethernet (RoCE), and Omni-Path Express (OPX). RDMA-enabled machines communicate with remote peers through a Queue Pair (QP), which includes a Send Queue and a Receive Queue. To send a message, a work queue element (wqe) is posted to the Send Queue via the user-level verbs API. Each QP is linked to a Completion Queue (CQ). Upon request completion, a completion queue element (cqe) is added to the CQ, which applications can poll to check the request status.

B. RDMA Transport Protocols

RDMA networks support various transport protocols, such as Reliable Connection (RC) and Unreliable Datagram (UD). RDMA includes two types of verbs: channel semantics and memory semantics. Channel semantics, or two-sided verbs, involve send and receive operations between a sender and a receiver. Memory semantics, or one-sided verbs, include RDMA read, write, and atomic operations, allowing direct memory access on a remote host without its involvement.

Table II summarizes features of RC and UD protocols. RC supports both one- and two-sided verbs with a large Maximum Transmission Units (MTU) of 2GB, offering high performance but limited scalability and a higher memory footprint. UD supports only basic send/receive operations with a 4KB MTU, requiring packetization for larger packets, but it scales better with constant resource consumption. Between RC and UD lies the dynamically connected (DC) transport service, which balances performance, one-sided verbs, and scalability by reducing the number of system-wide QPs through dynamic connections.

Figure 2 depicts the connection models of RC, UD and DC. Given a communication group of N nodes and PPN processes per node. To have a connection to any remote process, each process requires $((N-1) * PPN)$ QPs. Within a node, PPN processes share the same HCA, which means the HCA needs to handle $((N-1) * PPN^2)$ QPs. As a result, as the number of nodes increases, RC will have a scalability issue because of the explosion in the number of QPs needed as well as memory to store them. With these many QPs, the HCA cannot fit them all in its cache and will result in extra PCIe traffic to fetch them. For UD, a process only requires a QP to communicate with any remote processes, which means the number of QPs handled by the HCA is PPN and it remains constant as more nodes participate in the communication. Finally, for DC, it requires the same number of QPs as UD, but its connections are reliable while UD ones are not.

III. Desgin and Implemenation of OHIO

This section presents step by step a series of design choices from naive to high-performance and memory-optimized algorithms that are scalable and RDMA network agnostic.

A. Version 1: Naive Design

The key idea to achieve scalability is to **reduce the number of QPs used** during the Alltoall communication operation. In other words, we need to reduce the number of communicating pairs of processes to avoid cache thrashing happening with the NIC itself. In the very first design, a process leader within a node is responsible to exchange data of its local processes and itself to other peer leaders in other nodes. Figure 3 demonstrates the design which is comprised of two phases: (1) processes stage send data to shared memory region and (2) leader process posts send/receive requests to exchange data with remote leader processes while all processes copying data from shared memory to their receive buffers. Since the process leader cannot directly access data in the address space of the local processes, they are required to stage their data to a shared memory region. In this paper, it is referred to as shared send buffer (SSB). The leaders then initiate a data exchange with other leaders using Pairwise algorithm. Specifically, leader of node (i) communicate with the one of node $(i\, xor\, j)$ in which (j) is a communication step that runs

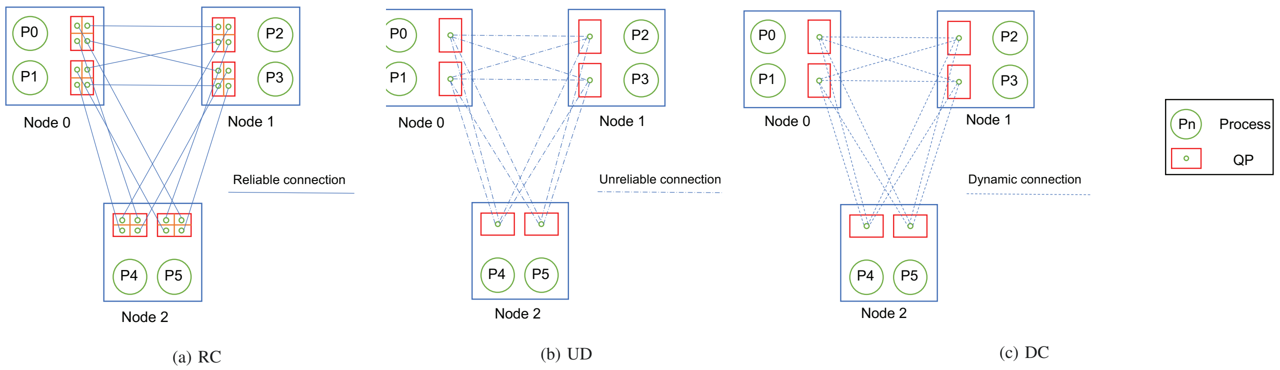


Fig. 2: Connection models for different transport protocols in RDMA networks. For RC, each process requires a dedicated QP for a remote connection. UD and DC can share a QP for connecting to any node. However, UD requires an extra software layer for reliability. Additionally, DC needs to tear down and establish a new connection when sending data to a different process.

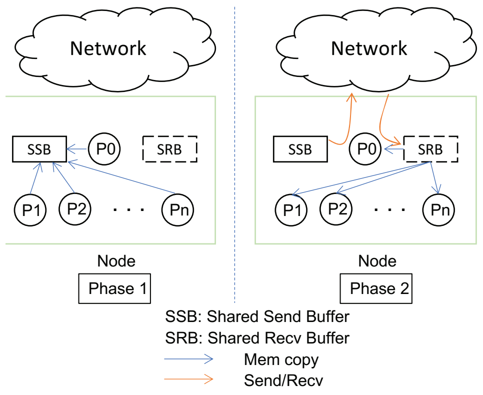


Fig. 3: Node level view of Naive Design with two phases: (1) stage data to SSB (2) send/receive operations to/from remote leader processes overlapped with memories copies to receive buffers from SRB.

from one to the number of nodes in a communication group (communicator) with power-of-two size. For the non-power-of-two case, in each communication step, the node leader transfers data to the right peer and receive from the left one. When the process leader exchanges data with a remote peer, it sends all of the data needed by that peer from the shared send buffer, receives the data from it and writes to another shared buffer, called shared receive buffer (SRB). While the send/receive operations progressed by HCA, processes can perform memory copies from the shared receive buffer to their receive buffers accordingly. The inter-node data transfers and memory copies are overlapped with each other in a pipelined fashion, i.e. data exchange with a current node overlapped with memory copies of data from the previous node.

Figure 4 depicts the changes in states of the shared send/receive buffers and receive buffers and how data are organized for the first design. In this example, send buffers are copied to shared send buffer in the order shown by the green zigzag. In other words, send buffers of process 0 and 1 (node 0) are written to the first and second halves of the shared send buffer. Note that shared send/receive buffers are contiguous one-dimensional arrays. In phase two, the purple data block consisting of four cells is sent to a remote leader process 3 (node 1) and written to its shared receive buffer in the same order as the shared send buffer of process 0. Because the data block is not contiguous, we need two send operations. Once the data arrive, process 2 and 3 copy out from the shared receive buffer and re-arrange to correctly place data in receive buffers, which results in the data block is being transposed.

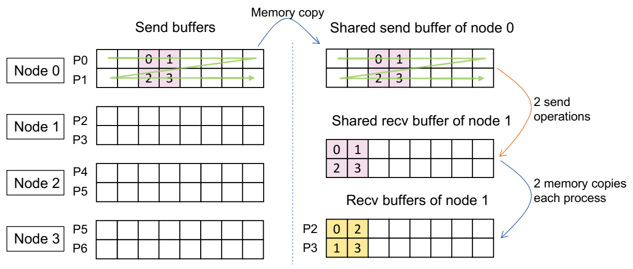


Fig. 4: State changes of the shared send/receive buffers and receive buffers for the first design. The green zigzags represent memory copy order from send buffer to the shared send buffer. Data order changes of a block are shown through changes in colors. Multiple send operations are required to communicate with a remote node. A data block is referred to as all the pieces of data needed by a remote node.

B. *Version 2: Performance Optimized Design*

Now that we can achieve high scalability by reducing the number of communicating pairs of processes through the leader-based scheme in the first design, the next step is to further optimize the performance. In the naive design, it takes multiple network operations to send all the data to a remote node due to the data not being contiguous as shown in Figure 4. Each message transfer is associated with a startup overhead which includes reading the work request element (wqe) stored in send queue of the corresponding QP. This can be an expensive operation, which leads QP cache thrashing if the QP is not currently stored in the NIC's cache. The HCA has to go through PCIe to access the main memory where the QP is stored. The second design focuses on optimizing the number of posted requests to HCA through **request coalescing**. Instead of invoking multiple send requests of small data size with multiple startup costs, when staging data to shared send buffer in phase (1), data are re-arranged so that it only takes a single send operation to communicate with a remote node.

Figure 5 describes in detail how data are manipulated and state changes in shared send/receive buffers and receive buffers. For the purple data block of four cells to be transferred to node 2 in a single network transfer, it is first copied to the shared send buffer from send buffers with re-arrangement in the order shown by the green zigzag, which requires multiple memory copies to re-order data. Since the data block is now contiguous, it can be directly placed into the shared receive buffer in a single send operation. After that, just like the naive design, once the data arrive, they can be copied out to receive buffers by individual processes. It is worthwhile to mention that even though we can transpose the data required by each node when copying data to the shared send buffer in Phase (1), which leads to a single network operation and a single memory copy by each process once data have arrived in shared receive buffer, this approach results in heavy strided memory access, which causes CPU cache thrashing when running a large scale.

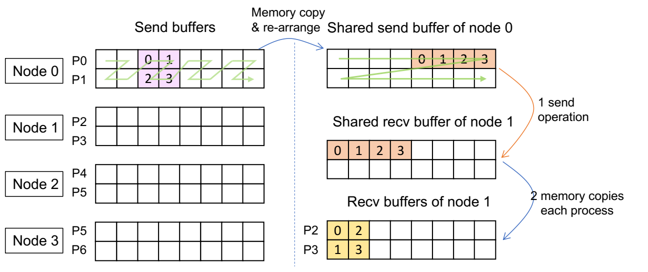


Fig. 5: State changes of the shared send buffers and receive buffers for the second design. The green zigzags represent the memory copy order from the send buffer to shared send buffer. Data order changes of a block are shown through changes in colors. Only a single send operation is required to communicate with a remote node. A data block is referred to as all the pieces of data needed by a remote node.

C. Version 3: Performance and Memory Optimized Design

The focus of the final design is to further reduce the memory footprint while retaining scalability and performance from the previous designs. The main idea behind the final design is **to reuse the space in shared receive buffer** for memory copy operations and network transfers in contrast to the second design which required extra shared memory to store all data received from other remote nodes. Figure 6 demonstrates the design idea in which the shared receive buffer only requires the space to contain two blocks of data. A data block in this paper is referred to all the pieces of data needed by a remote node. In Figure 6, when exchanging data with node (N), the first half of the shared receive buffer is used to receive the data block from node (N) while the second half of the shared buffer contains the data block from node $(N-1)$ that is copied to receive buffers. In the next communicating step to node $(N+1)$, the roles of these shared regions are swapped. The second region contains the data from receive operations while the first region is now used for memory copy. Pseudocode of the design is presented in Algorithm 1.

D. Analysis of Proposed Designs

Table III summarizes the discrepancies between existing algorithms and the proposed designs on various metrics, QPs

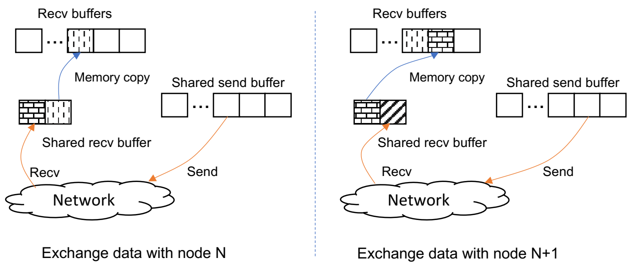


Fig. 6: Performance and Memory Optimized Design: in contrast to the second design which required extra shared memory to store all receive data from other leader processes, in this design, the shared receive buffer only requires the space to contain two blocks of data. One block is for memory copy and another is for send operation. A data block is referred to as all the pieces of data needed by a remote node.

Algorithm 1: Performance and Memory Optimized

```
1  N ← number of nodes
2  PPN ← processes per nodes
3  M ← message size
4  lrank ← local rank in a node
   /* Get shared buffers */
5  shm_recv_buf ← $get\_shared\_buf()$
6  shm_send_buf ← $get\_shared\_buf()$
   /* Copy send buf to shared send buf in the
      order shown in Fig. 5 */
7  for $i \leftarrow 0$ to $N-1$ do
8      src_add ← $addr\_to\_copy\_from\_send\_buf$
9      dst_add ← $addr\_to\_copy\_to\_shm\_send\_buf$
10     $mem\_copy(dst\_add,\ src\_add,\ M * PPN)$
11 $MPI\_Barrier(node\_comm)$
12 for $i \leftarrow 0$ to $N$ do
       /* Process leaders do pairwise exchange */
13     if $i < N$ AND $lrank = 0$ then
           /* Pairwise pattern */
14         src ← dst ← $node\_num \oplus i$
           /* Send buf to node 'src' */
15         tmp_send ←
             $addr\_to\_send\_from\_shm\_send\_buf$
           /* Recv buf from node 'dst' */
16         tmp_recv ←
             $addr\_to\_recv\_from\_shm_recv\_buf$
17         $MPI\_Irecv(tmp\_recv,\ src)$
18         $MPI\_Isend(tmp\_send,\ dst)$
       /* Copy from shared recv buf to recv buf */
19     for $j \leftarrow 0$ to $PPN-1$ do
20         src_addr ←
             $address\_to\_copy\_from\_shmem\_recv\_buf$
             dst_addr ← $address\_to\_copy\_to\_recv\_buf$
             $mem\_copy(dst\_addr,\ src\_addr,\ M)$
21     $MPI\_Barrier(node\_comm)$
```

usage, the number of posted requests to HCA, and extra memory footprint. Given a communicator of N nodes and PPN processes per node doing an Alltoall of message size M, the

TABLE III: Analysis of Proposed Designs for Alltoall Communication

Algorithm	**Number of QPs used per node**	**Number of posted send requests per node**	**Extra memory footprint**
Proposed Naive Design (V1)	$(N-1)$	$(N-1)*PPN$	$2*M*N*PPN^2$
Proposed Performance Optimized Design (V2)	$(N-1)$	$(N-1)$	$2*M*N*PPN^2$
Final Proposed Performance and Memory Optimized Design (V3)	$(N-1)$	$(N-1)$	$(M+2)*N*PPN^2$

TABLE IV: Hardware Characteristics of Clusters Used in Experiments

Hardware	**Cluster A**	**Cluster B**
Processors	Intel 8280 “Cascade Lake”	Intel Xeon Platinum 8380 ”Ice Lake”
Cores/Node	56 (28 per socket)	80 (40 cores/socket) 2.3 GHz
Clock Rate	2.7Ghz	2.3 GHz
Memory/Node	192GB DDR-4	256GB (3.2 GHz) DDR4
Network	Mellanox Infiniband, HDR-100	Omni-Path, 100Gb/sec

numbers of QPs used and the numbers of posted requests to HCA significantly affect the performance and scalability of an algorithm. The final proposed design only requires $(N-1)$ QPs as well as the number of posted requests, which are quite small compared to other algorithms like Pairwise and Scatter Destination. The higher the QPs usage, the more memory is required and the greater chance that NIC’s cache cannot store all of the QPs for communication, which leads to QP cache thrashing. As a result, they suffer performance degradation and are not able to scale well. In terms of memory footprint, we trade off some extra memory to reduce the number of QPs and posted requests. The final design leaves the least memory footprint among the three proposed designs.

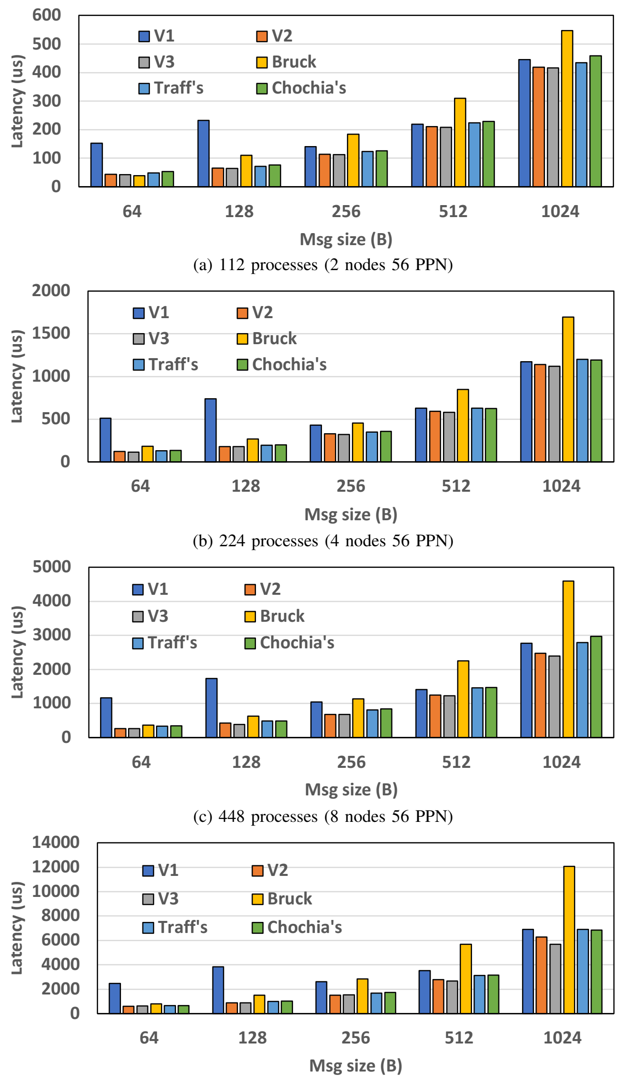
IV. Evaluation

A. Experimental Setup

We conduct all the experiments on two clusters with different interconnects, InfiniBand and Omni-Path, to support the claim that the proposed designs are network-agnostic. Table IV details the hardware characteristics of the two clusters. The efficiency of the proposed designs is evaluated against all major MPI implementations, open and closed source, which have optimized designs being continuously developed in their codebases. Specifically, we compare with MVAPICH2-X version 2.3 [21], Intel MPI Version 2021.7 Build 20221022 [22], Open MPI 5.0.1a1 [23] + UCX version 1.15.0 [23], and HPCX 2.13 [24]. The proposed designs are evaluated at both micro-benchmark and application levels. We use OSU Micro-Benchmarks 7.1 (OMB), which is widely adopted by both academic and industrial communities for benchmarking performance. To be certain that results are reproducible, all of the experiments are run 5 times to remove any noise or fluctuation. Within each OMB run, each message is an average of 1000 iterations for small size and 100 iterations for large size.

B. Performance Evaluation of Micro-benchmark

The three proposed designs, hereafter referred to as V1, V2 and V3, are first evaluated against Bruck’s and two state-of-the-art hierarchical algorithms: Traff’s [15] and Chochia’s [16]



(a) 112 processes (2 nodes 56 PPN)

(b) 224 processes (4 nodes 56 PPN)

(c) 448 processes (8 nodes 56 PPN)

(d) 896 processes (16 nodes 56 PPN)

Fig. 7: Performance Evaluation of the Proposed Designs (V1, V2 and V3) with Existing Algorithms using RC on Cluster A (InfiniBand)

on Cluster A with InfiniBand adapters. Figure 7 demonstrates the performance of the proposed solutions for an Alltoall operation of 112, 224, 448 and 896 processes. Compared to V1, naive design, V2 and V3 significantly outperform it as the latter designs reduce the number of requests posted to the HCA through the request coalescing technique. The degree of improvement becomes larger as the process count increases. Between V2 and V3, the final design (V3) performs better than V2 as message size and process count increase. This is attributed to the size optimization of the shared receive buffer. The smaller the buffer is, the better it can fit into CPU’s cache. Finally, the final design, V3, outperforms both flat design (Bruck’s) and state-of-the-art hierarchical design (Traff’s and Chochio’s), exhibiting an average latency improvement of 40%-50% and 9%-15%, respectively.

Figure 8 compares the numbers of PCIe read transactions of the proposed design with Intel MPI and MVAPICH2-X. We use Intel Performance Counter Monitor to gather these performance numbers on our local clusters where we have root access. The cluster has 8 nodes, equipped with Intel Xeon Gold 6132 CPU @ 2.60GHz 28 cores per node and Connect-X5 adapters (100 Gb/sec 4X EDR). The proposed design uses less PCIe reads than the two default MPI libraries in the cluster, demonstrating the alleviation of unneeded QP traffic.

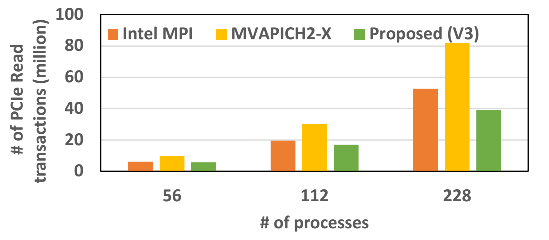
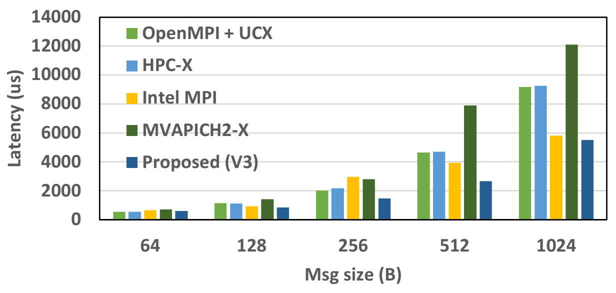


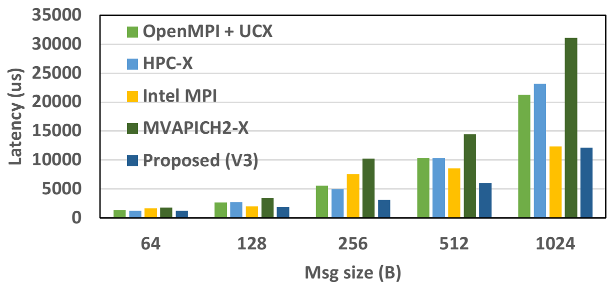
Fig. 8: Evaluation of the numbers PCIe read transactions

Now that the above experiment shows that the final design, V3, delivers the best performance among the proposed ones, V3 will be compared with other MPI implementations. Figure 9 demonstrates the efficiency of the proposed design (V3) compared with Open MPI + UCX, HPC-X, Intel MPI and MVAPICH2-X on Cluster A with InfiniBand interconnect when performing Alltoall communication with 896, 1792, 3584 and 7168 processes. Overall, the proposed design outperforms all MPI implementations evaluated in this experiment. **On the largest scale with 7168 processes, the proposed design performs 61.13%, 56.40%, 37.49% and 51.90% on average better than Open MPI + UCX, HPC-X, Intel MPI and MVAPICH2-X on InfiniBand cluster, respectively.**

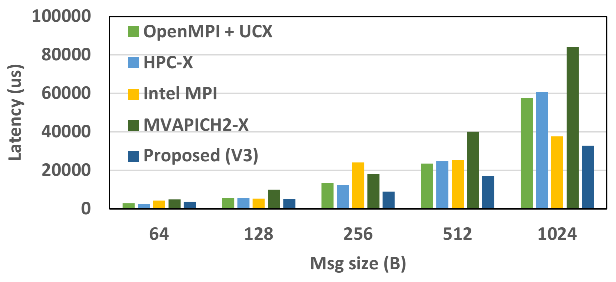
To demonstrate the network-agnostic property of the proposed design, we also evaluate it on Cluster B with Omni-Path network and Intel Ice Lake architecture of 80 cores per node. In summary of the results presented in Figure 10, we observe the same trend as with Cluster A (InfiniBand). On 1280 and 2560 processes, the proposed design performs better than other MPI implementations and competitively to Intel MPI. When processor count per node increases from 56 (Cluster A) to 80 (Cluster B), the number of QPs required per node for communication also increases. We observe some jump in latency at 512 and 1024 bytes for Open MPI + UCX. In addition, also at these message points, HPC-X is unable to run due to QP errors output by the library. This is expected as Alltoall is performed at large scale because we may be running out of QPs and memory if Alltoall algorithms are not QP-aware. Finally, the scalability of the design is evaluated by comparing with different transport protocols, namely RC, DC, and UD, ranging from the most high-performance and least scalable to the opposite. In this experiment, we use the MVAPICH2-X library as it has support for all three transport protocols. Figure 11 demonstrates the scalability of the design for Alltoall communication of 512 and 1024 bytes on 2 to 128 nodes (from 112 to 7168 process). **When compared with the tuned algorithm in MVAPICH2-X that picks the best transport among the three, RC, DC and UD depending on message size and process count, the proposed design outperforms by up 65.2% and 69.5% for 512 and 1024 byte Alltoall, respectively.**



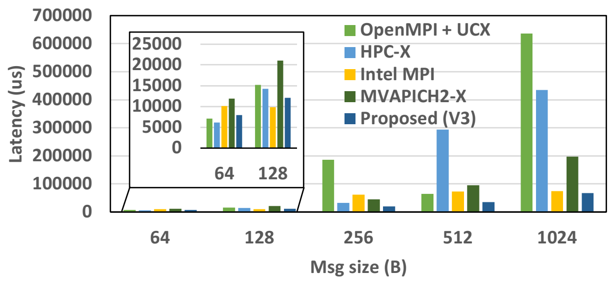
(a) 896 processes (16 nodes 56 PPN)



(b) 1792 processes (32 nodes 56 PPN)



(c) 3584 processes (64 nodes 56 PPN)

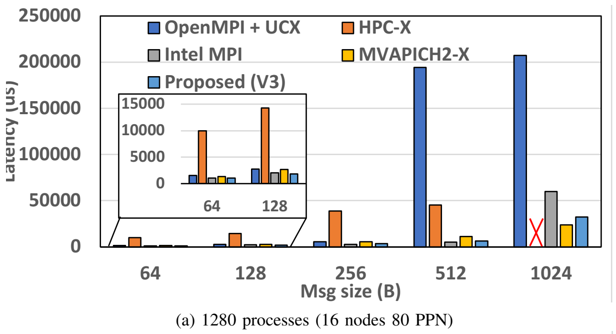


(d) 7168 processes (128 nodes 56 PPN)

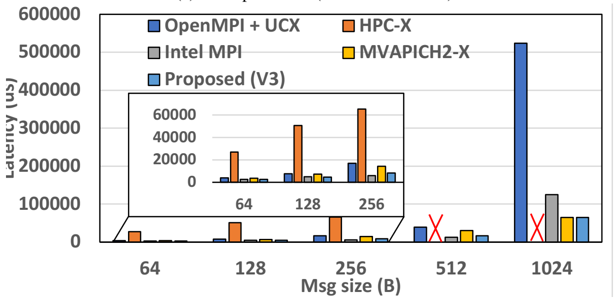
Fig. 9: Performance Evaluation of the Proposed Design (V3) with MPI libraries on Cluster A (InfiniBand).

C. Performance Evaluation of CPMD Application

Car-Parrinello Molecular Dynamics (CPMD) [2] is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, which is particularly designed for ab-initio molecular dynamics. It brings together methods including classical molecular dynamics, solid-state physics and quantum chemistry. **The application has undergone thorough**

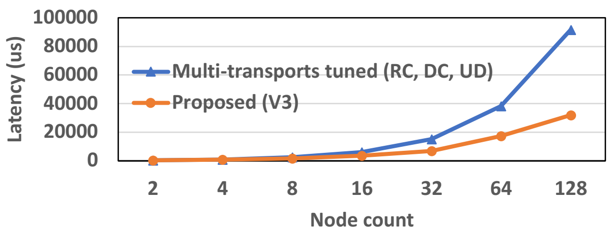


(a) 1280 processes (16 nodes 80 PPN)

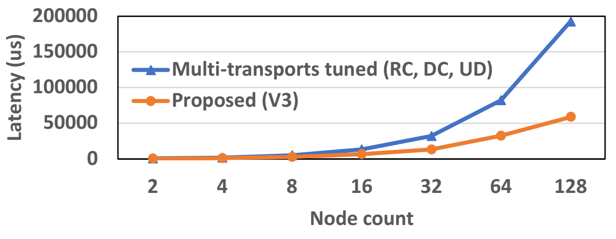


(b) 2560 processes (32 nodes 80 PPN)

Fig. 10: Performance Evaluation of the Proposed Design (V3) with MPI libraries on Cluster B (Omni-Path).



(a) 512 Byte Message Size



(b) 1024 Byte Message Size

Fig. 11: Scalability Evaluation of the Proposed Design with Multiple Transport Protocols on Cluster A (InfiniBand) from 2 to 128 nodes with 56 PPN.

analysis in the best practices documented by the HPC Advisory Council [25]. Alltoall, being the predominant communication pattern, is heavily utilized in the simulation. Additionally, the message size falls within the small range. Consequently, the performance of CPMD is closely dependent on the efficiency of Alltoall operations. In this experiment, CPMD is evaluated with Carbon-120-inp2, an input set that is often used for testing its performance. Figure 12 demonstrates the performance improvement of the proposed solution (V3) compared to MVAPICH2-X, Open MPI + UCX, and Intel MPI on 160, 320 and 640 processes. We were unable to compare with HPC-X due to the mismatch between the GNU versions used to compile HPC-X and CPMD. HPC-X is a prebuilt package using 4.8.5 while CPMD requires a later version, 7.1.0 in this case. In summary, **the proposed design (V3) outperforms MVAPICH2-X, Open MPI + UCX, and Intel MPI by at most 24.98 %, 40.44% and 50.48%, respectively in the simulation time**.

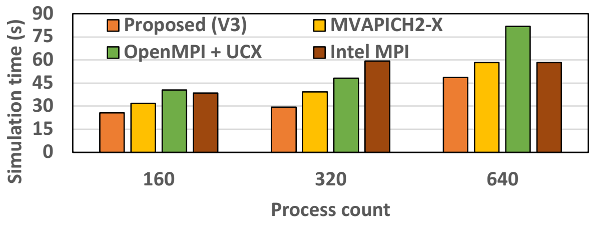


Fig. 12: Performance Evaluation of CPMD Application on Cluster B (Omni-Path), ranging from 160 to 640 Processes.

V. Discussion

Design Applicability to Other Collectives and Variants: The design ideas presented in this paper are general and can be expanded to encompass non-blocking collectives as well as other variants such as MPI_Ialltoall, MPI_Alltoallv, and MPI_Alltoallw. In the majority of open-source contemporary MPI libraries (MPICH, OpenMPI, MVAPICH) blocking and non-blocking Alltoall collectives utilize the same communication mechanisms underneath to transfer data. So, although new implementations will be required, the design principles should be applicable to MPI_Ialltoall. In addition, the current designs can certainly be extended for MPI_Alltoallv and MPI_Alltoallw to account for the distinct workload required by each process.

Mitigating QP Explosion with MPI+X: MPI+X, where X can be OpenMP, is widely considered a good alternative to pure MPI-based applications. This approach tackles the QP issue by ensuring that there is only one MPI process per NUMA (Non-Uniform Memory Access) domain or node. As a result, the number of QPs needed is minimized, resembling the approach proposed in the designs. Nevertheless, there are still hundreds of applications out there which cannot or do not use OpenMP due to various performance and functionality reasons. All of these would require application redesign which most application developers are hesitant to do. Our technique, on the other hand, is application-agnostic and requires no application-level changes. Various publications, notably [26], have also studied this and demonstrated that collective communication is still heavily used. Thus, our proposed approach at the middleware level will still be relevant.

Adaptability of Designs for Diverse High-Performance Networks: Most modern interconnects have similar concepts of communication end-points such as Queue Pairs. In this paper, InfiniBand and Omni-Path are used as case studies for the network-agnostic property of the presented designs. We believe similar issues and challenges will exist in other high-performance networks such as RoCE, making the designs applicable.

VI. Related Work

Algorithmic Optimization of Alltoall with/without Network Support: In [27], the authors improved Bruck's [13] by proposing a different way of shifting data blocks in the first phase that eliminates the need to do data rotations in the last phase. Additionally, by using derived datatype operations such as pack and unpack, a zero-copy Bruck can be implemented without any explicit memory copies for data re-organizations. However, this optimization is not a true zero-copy. Inside of pack and unpack primitives provided by MPI, there are memory operations invoked to pack data to temporary buffers before sending or unpack data after receiving.

Xu et al. [28] proposed novel data structures for optimizing the performance of Alltoallv. The proposed structures can be used to avoid local memory rotation and data movement. The authors also proposed a hierarchical design to take advantage of process locality. Fan et al. [29] pointed out the weaknesses of [28] and presented their own implementations of Alltoall and Alltoallv. Prisacari et al. [30] optimized Alltoall on hierarchical networks, namely fat tree and dragonfly, for the case of non-power-of-two processes where XOR communication pattern cannot be applied. Venkata et al. [31] utilized CORE-Direct to remove local data shifting and rotation performed by memory copy operations in Bruck's algorithm. CORE-Direct is a network feature of InfiniBand that allows offloading collectives to network adapters. It also allows to send data stored in different buffers in a single network operation.

Li et al. [32] proposed cache-oblivious algorithms for Alltoall. Their work primarily focuses on optimizing for the CPU cache, whereas our designs address the challenges of QP-trashing in the NIC cache. Additionally, they require a shared heap to see benefits, while our approach does not impose such a requirement. [17] utilized shared memory in multi-core systems to have better intra-node latency; in the inter-node phase, all processes within a node exchange data with remote ones, which increases the QP usage per node. Our work leverages shared memory to stage data, enabling the node leader to exchange data on behalf of others, which reduces QP usage and mitigates NIC cache thrashing for better scalability.

RDMA Network Optimization: Many studies have focused on optimizing RDMA networks. Kalia et al. [6] provided design guidelines to enhance RDMA performance by reducing PCIe transactions and considering NIC architecture. Monga et al. [11] developed a scalable RPC system with RC-based QP sharing to reduce QP cache thrashing and CPU overhead. Bae et al. [20] introduced RDMA Box, including techniques like request chaining/merging and adaptive polling to minimize handshake overhead and load to NIC, and reduce CPU usage.

Friedley et al. [33] proposed using UD to address QP memory and scalability issues in MPI. Although UD is unreliable and requires a software layer for reliability and fragmentation, their results showed UD's latency and bandwidth without the reliability layer on top are comparable to RC. In contrast, our approach retains the feature-rich RC and resolves QP issues without the additional overhead of UD.

VII. Conclusion and Future Work

In this paper, three QP-aware designs are proposed to address QP issues, from the naive to performance and memory-optimized ones. Various techniques and insights that are previously studied for RDMA network optimizations are applied These designs are RDMA network agnostic and can be applied to any high-speed network The performance of the final design is evaluated with different MPI implementations. Its scalability is also assessed with different transport protocols. In recapitulation, the final proposed design provides 61.13%, 56.40%, 37.49% and 51.90% on average better than Open MPI + UCX, HPC-X, Intel MPI and MVAPICH2-X at micro-benchmark level with up to 7168 cores, respectively. In addition, the evaluation at the application level with Car-Parrinello Molecular Dynamics code shows 24.98%, 40.44% and 50.48% improvement in the simulation time, compared to MVAPICH2-X, Open MPI + UCX, and Intel MPI, respectively. In the future, we plan to design a kernel-assisted solution that can help eliminate the shared memory portion and encompass large messages, resulting in a complete solution with high-performance, good scalability and low memory footprint.

Acknowledgment

This research is supported in part by NSF grants #1818253, #1854828, #2007991, #2018627, #2112606, #2311830, #2312927, and XRAC grant #NCR-130002.

References

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

[2] J. Hutter and M. Iannuzzi, "CPMD: Car-parrinello molecular dynamics," *Zeitschrift für Kristallographie-Crystalline Materials*, vol. 220, no. 5-6, pp. 549–551, 2005.

[3] D. Pekurovsky, "P3DFFT: A framework for parallel computations of fourier transforms in three dimensions," *SIAM Journal on Scientific Computing*, vol. 34, no. 4, pp. C192–C209, 2012.

[4] J. A. Yang, J. Park, S. Sridharan, and P. T. P. Tang, "Training deep learning recommendation model with quantized collective communications," in *Conference on Knowledge Discovery and Data Mining (KDD)*, 2020.

[5] D. Kalamkar, E. Georganas, S. Srinivasan, J. Chen, M. Shiryaev, and A. Heinecke, "Optimizing deep learning recommender systems training on cpu cluster architectures," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–15.

[6] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance RDMA systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 437–450.

[7] Z. Wang, L. Luo, Q. Ning, C. Zeng, W. Li, X. Wan, P. Xie, T. Feng, K. Cheng, X. Geng *et al.*, "SRNIC: A scalable architecture for RDMA NICs," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1–14.

[8] X. Kong, Y. Zhu, H. Zhou, Z. Jiang, J. Ye, C. Guo, and D. Zhuo, "Collie: Finding performance anomalies in RDMA subsystems," in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 287–305.

[9] A. Kalia, M. Kaminsky, and D. Andersen, "Datacenter RPCs can be general and fast," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 1–16.

[10] Y. Chen, Y. Lu, and J. Shu, "Scalable RDMA RPC on reliable connection with efficient resource sharing," in *Proceedings of the Fourteenth EuroSys Conference 2019*, 2019, pp. 1–14.

[11] S. K. Monga, S. Kashyap, and C. Min, "Birds of a feather flock together: Scaling RDMA RPCs with flock," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 212–227.

[12] H. Subramoni, K. Hamidouche, A. Venkatesh, S. Chakraborty, and D. K. Panda, "Designing MPI library with dynamic connected transport (DCT) of infiniband: early experiences," in *Supercomputing: 29th International Conference, ISC 2014, Leipzig, Germany, June 22-26, 2014. Proceedings 29*. Springer, 2014, pp. 278–295.

[13] J. Bruck, C.-T. Ho, S. Kipnis, and D. Weathersby, "Efficient algorithms for all-to-all communications in multi-port message-passing systems," in *Proceedings of the sixth annual ACM symposium on Parallel algorithms and architectures*, 1994, pp. 298–309.

[14] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in MPICH," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[15] J. L. Träff and A. Rougier, "MPI collectives and datatypes for hierarchical all-to-all communication," in *Proceedings of the 21st European MPI Users' Group Meeting*, 2014, pp. 27–32.

[16] G. Chochia, D. Solt, and J. Hursey, "Applying on node aggregation methods to MPI alltoall collectives: Matrix block aggregation algorithm," in *Proceedings of the 29th European MPI Users' Group Meeting*, 2022, pp. 11–17.

[17] R. Kumar, A. Mamidala, and D. K. Panda, "Scaling alltoall collective on multi-core systems," in *2008 IEEE International Symposium on Parallel and Distributed Processing*. IEEE, 2008, pp. 1–8.

[18] L. A. Steffenel, "Modeling network contention effects on all-to-all operations," in *2006 IEEE International Conference on Cluster Computing*. IEEE, 2006, pp. 1–10.

[19] K. S. Khorassani, C.-H. Chu, Q. G. Anthony, H. Subramoni, and D. K. Panda, "Adaptive and hierarchical large message all-to-all communication algorithms for large-scale dense gpu systems," in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. IEEE, 2021, pp. 113–122.

[20] J. Bae, L. Liu, Y. Wu, G. Su, and A. Iyengar, "RDMAbox: Optimizing RDMA for memory intensive workload," in *2021 IEEE 7th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2021, pp. 1–10.

[21] (2023) MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE. http://mvapich.cse.ohio-state.edu/. Accessed Mar 15, 2023.

[22] (2023) Intel® MPI library deliver flexible, efficient, and scalable cluster messaging. https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html. Accessed Mar 15, 2023.

[23] (2023) Open MPI: Open source high performance computing. https://www.open-mpi.org/. Accessed Mar 15, 2023.

[24] (2023) Hpc-x. https://developer.nvidia.com/networking/hpc-x. Accessed Mar 15, 2023.

[25] (2023) Hpc advisory council best practices. https://www.hpcadvisorycouncil.com/best_practices.php, Accessed Mar 15, 2023.

[26] I. Laguna, R. Marshall, K. Mohror, M. Ruefenacht, A. Skjellum, and N. Sultana, "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*, 2019, pp. 1–14.

[27] J. L. Träff, A. Rougier, and S. Hunold, "Implementing a classic: Zero-copy all-to-all communication with MPI datatypes," in *Proceedings of the 28th ACM international conference on Supercomputing*, 2014, pp. 135–144.

[28] C. Xu, M. G. Venkata, R. L. Graham, Y. Wang, Z. Liu, and W. Yu, "Sloavx: Scalable logarithmic alltoallv algorithm for hierarchical multicore systems," in *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. IEEE, 2013, pp. 369–376.

[29] K. Fan, T. Gilray, V. Pascucci, X. Huang, K. Micinski, and S. Kumar, "Optimizing the bruck algorithm for non-uniform all-to-all communication," in *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing*, 2022, pp. 172–184.

[30] B. Prisacari, G. Rodriguez, and C. Minkenberg, "Generalized hierarchical all-to-all exchange patterns," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. IEEE, 2013, pp. 537–547.

[31] M. G. Venkata, R. L. Graham, J. Ladd, and P. Shamis, "Exploring the all-to-all collective optimization space with connectx core-direct," in *2012 41st International Conference on Parallel Processing*. IEEE, 2012, pp. 289–298.

[32] S. Li, Y. Zhang, and T. Hoefler, "Cache-oblivious MPI all-to-all communications based on morton order," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 542–555, 2017.

[33] A. Friedley, T. Hoefler, M. L. Leininger, and A. Lumsdaine, "Scalable high performance message passing over infiniband for open MPI," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2007.