2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW) | 978-1-6654-9747-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPSW55747.2022.00014

Highly Efficient Alltoall and Alltoallv Communication Algorithms for GPU Systems

Chen-Chun Chen*, Kawthar Shafie Khorassani*, Quentin G. Anthony*, Aamir Shafi*,
Hari Subramoni† and Dhabaleswar K. Panda†
Department of Computer Science and Engineering
The Ohio State University, Columbus, Ohio
*Email: {chen.10252, shafiekhorassani.1, anthony.301, shafi.16}@osu.edu
†Email: {subramon, panda}@cse.ohio-state.edu

***Abstract*—In recent years, High Performance Computing (HPC) and Deep Learning (DL) applications have been modified to run on top supercomputers and utilize the high compute power of GPUs. While GPUs provide high computational power, communication of data between GPUs and across a network continues to be a bottleneck. In particular, with the increasing amount of FFT compute and sparse matrix transpose operations in these applications, Alltoall MPI collective operations are heavily used. Alltoall communication is considered the heaviest communication pattern compared to other MPI collective calls. Few techniques and algorithms effectively help in optimizing Alltoall communication, much less improving the performance on a dense GPU cluster while exploiting the features of modern interconnects and topologies. Despite the introduction of Inter-Process Communication (IPC) in CUDA 4.1 by NVIDIA, state-of-the-art MPI libraries have not utilized these IPC-based mechanisms to design novel Alltoall algorithms that exploit the capabilities of modern GPUs.**

In this paper, we propose hybrid IPC-advanced designs for Alltoall and Alltoallv communication on novel GPU systems. By utilizing zero-copy load-store IPC mechanisms for multi-GPU communication within a node, we are able to overlap the intra-node and inter-node communication, yielding improved performance on GPU systems.

We evaluate the benefits of our designs at the benchmark and application layers on the ThetaGPU system at ALCF and the Lassen system at LLNL. Our designs provide up to 13.5x and 71% improvements on 128 GPUs and 64 GPUs at the benchmark-level over state-of-the-art MPI libraries on ThetaGPU and Lassen respectively. At the application level, our designs have up to 59x performance improvement for an HPC application, heFFTe, and 5.7x performance improvement for a Deep Learning application, DeepSpeed, on 64 GPUs on ThetaGPU and 256 GPUs on Lassen.

***Index Terms*—MPI, GPU, DGX, IPC, Alltoall, Alltoallv**

I. Introduction and Motivation

Alltoall communication is widely used in many HPC and scientific applications and DL frameworks. It allows data in each process to be transferred to every process, which results in the most communication-heavy pattern over other MPI collective operations. For example, in an application run with n processes, Alltoall communication requires $O(n^2)$ communication for data transfer. As the scale and the message size increase, the heavier communication load leads to higher overall latency across the network. Due to the heavy communication workload, few algorithms or techniques are really efficient for improving communication performance. Recently, more applications use GPUs to accelerate computation, and the data is often stored in device memory, which increases the communication cost regardless of whether it is in an intra-node or an inter-node environment.

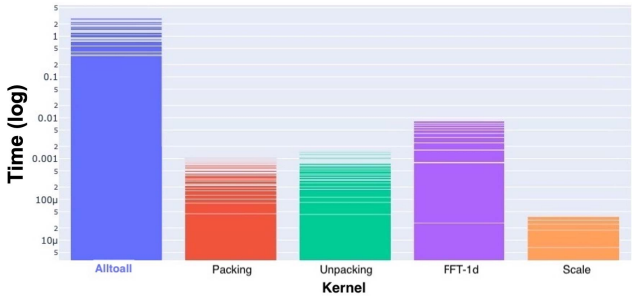


Fig. 1. The bottleneck of MPI_Alltoall in the heFFTe application on 24 GPUs (Courtesy [1])

For instance, heFFTe [2] is a popular library, with support for modern GPU-based HPC systems, that provides efficient and scalable implementations of the Fast Fourier Transform (FFT) algorithm—a widely used computational kernel in many HPC applications. Figure 1 depicts an overall execution time profile of a typical heFFTe application run. The profile shows the time spent in Alltoall in comparison to the time spent in other application tasks including computing FFTs and packing and unpacking kernels. It is clear that the MPI_Alltoall operation is the main bottleneck for the heFFTe application. In this context, it is critical to optimize the performance of Alltoall communication at the MPI layer in order to improve the performance of HPC applications like heFFTe.

Historically, the computation nodes of GPU-based HPC systems were equipped with fewer GPUs than nodes on modern cluster deployments. This meant that the overall communication performance of GPU applications was bound by the bandwidth of the network during multi-node data transfers and bound by the bandwidth of the inter-GPU interconnects for intra-node data transfers. However with the emergence of modern architectures and systems, a single node is equipped with more GPUs, and nodes are connected with a faster network. One such example is the NVIDIA DGX-A100 system illustrated in Figure 2. It is powered by 8

978-1-6654-9747-3/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPSW55747.2022.00014

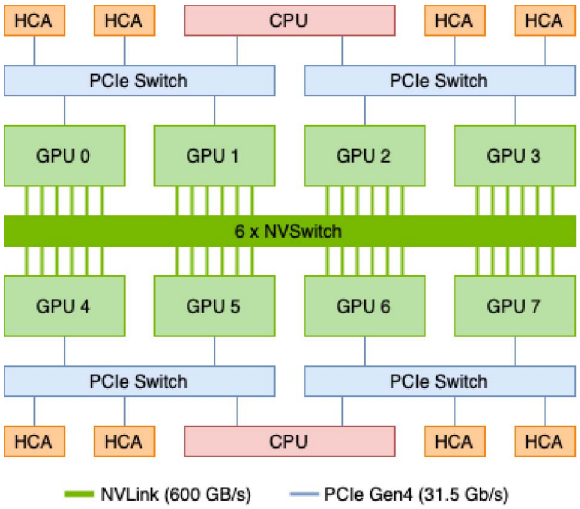


Fig. 2. Topology of DGX-A100 Node

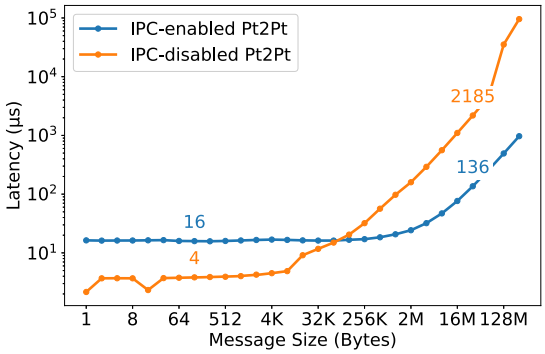


Fig. 3. Comparison of IPC enabled and IPC disabled in a point-to-point latency benchmark (OMB) on a DGX-A100 system

A100 Tensor Core GPUs per node, and fully connected by NVLink/NVSwitch technology. In this system, each socket contains 2 PCIe switches, and each PCIe switch connects with 2 Mellanox NICs. This powerful connection not only improves the communication between devices within a node through NVLink, but lowers the latency between GPUs across nodes through the presence of multiple NICs. In particular, it enhances the performance of a scenario allowing multiple GPUs to transfer data simultaneously. These novel architectures attract the users to run their applications on a larger scale and with a heavier workload, which makes the optimization for Alltoall communication even more critical.

While IPC has been utilized for point-to-point communication to enhance the performance of medium to large message sizes, as depicted in Figure 3, it has yet to be extended widely to GPU-aware collective designs. **State-of-the-art MPI libraries do not utilize IPC-based designs for Alltoall collectives in order to enhance the inter-GPU communication through the NVLinks available on many modern dense GPU systems.** Hence, we propose IPC-advanced Alltoall communication algorithms for improving Alltoall and Alltoallv communication on novel dense GPU systems. The heavily connected GPUs within a node motivates the usage of IPC for collective operations in order to allow for an overlap of intra-node communication through IPC and inter-node communication with other protocols. We move data in IPC buffers by launching kernels, referred to as *kernel-based IPC* for medium sizes and utilize CUDA memcpy APIs to transfer data, referred to as *Memcpy-based IPC*, for large message sizes. Figure 4 depicts the performance of optimized Alltoall design (with existing Alltoall algorithms and techniques) compared to Memcpy-based IPC and kernel-based IPC on 8 GPUs on a DGX-A100 node.

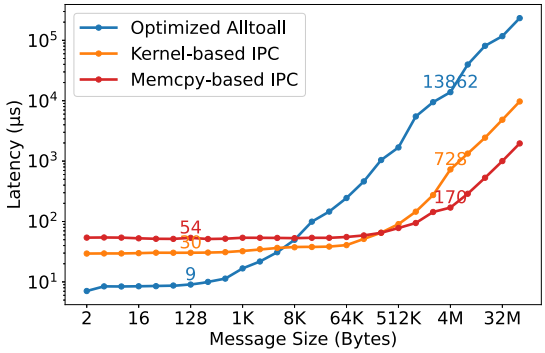


Fig. 4. Latency comparison of optimized Alltoall design (with existing Alltoall algorithms and techniques), proposed kernel-based IPC, and Memcpy-based IPC designs for an MPI_Alltoall benchmark (OMB) on 8 GPUs on a DGX-A100 system

Besides Alltoall, Alltoallv communication is also widely applied by applications since it allows arbitrary send (recv) counts to be transferred between any 2 processes. For example, heFFTe supports data transferring with both Alltoall and Alltoallv communication. This further motivates the need to not only apply optimized designs for Alltoall but also to evaluate the benefits of extending these novel designs to Alltoallv communication patterns as well.

A. Challenges

We address the following challenges to develop efficient GPU-based Alltoall and Alltoallv algorithms using hybrid IPC-advanced techniques for modern dense GPU systems:

- Can we utilize the features, e.g.: NVLink, provided by heavily connected GPUs on modern systems to motivate the need for optimized MPI collective algorithms that fully exploit this high connectivity?
- How can we exploit the zero-copy load-store mechanisms of IPC to design GPU-aware IPC-advanced Alltoall and Alltoallv collective operations?
- What are the limitations within existing communication libraries with respect to running and optimizing the MPI bottleneck of certain workloads?

- Can we optimize the performance of HPC and DL workloads that are reliant on Alltoall and Alltoallv for FFT and matrix transpose using optimized designs for MPI_Alltoall and MPI_Alltoallv to provide enhanced performance and scalability?

B. Contributions

In this paper, we design hybrid IPC-advanced Alltoall and Alltoallv communication algorithms to optimize the performance of GPU-aware MPI_Alltoall and MPI_Alltoallv across different dense GPU platforms.
This paper makes the following contributions:

1) Identify challenges with existing MPI_Alltoall and MPI_Alltoallv implementations and utilizing existing architectures to motivate the need for an optimized MPI_Alltoall and MPI_Alltoallv algorithm that exploits the available interconnect and technology.
2) Propose a kernel-based and memcpy-based IPC-advanced algorithm implementation of GPU-based MPI_Alltoall and MPI_Alltoallv.
3) Implement the GPU-aware IPC-advanced algorithm and propose hybrid designs. The integrated designs support single/multi Alltoall and Alltoallv communication patterns and cover common MPI datatypes, which benefits a variety of use cases required by certain applications.
4) Develop a comprehensive performance evaluation of GPU-aware MPI_Alltoall and MPI_Alltoallv using the proposed designs compared to the state-of-the-art GPU-aware communication libraries (i.e. MVAPICH2-GDR, NCCL, IBM Spectrum MPI, and OpenMPI+UCX) at the benchmark level, using OSU-Microbenchmarks and at the application level, using heFFTe, PSDNS, DeepSpeed, and DLRM on a DGX-A100 system (ThetaGPU) and a Power9 V-100 system (Lassen).

To the best of our knowledge, this is the first paper focusing on developing hybrid designs using IPC- and GPU-based MPI collective algorithms for Alltoall and Alltoallv communication.

II. Background

A. GPU-aware MPI

The Message Passing Interface (MPI) is a standard for exchanging messages on parallel and distributed architectures. State-of-the-art *GPU-aware* communication libraries [3], such as MVAPICH2-GDR [4], OpenMPI [5], IBM Spectrum MPI [6], and NCCL [7] provide communication primitives optimized for GPUs and networking. Modern GPU-Aware MPI libraries employ a collection of optimizations such as GPUDirect RDMA, CUDA Inter-Process Communication (IPC), and pipelining mechanisms.

B. Inter-Process Communication (IPC)

Since CUDA 4.1, the Inter-Process Communication (IPC) interface has enabled the efficient transfer of messages between GPUs within the same node. Specifically, CUDA IPC enables any process to share its GPU device buffer with any other remote node-local process. Once the GPU buffer is shared, the remote process can map this device buffer into its own address space and issue CUDA transfer operations such as `cuMemcpy` to it directly. An MPI library can use CUDA IPC to efficiently transfer GPU data within a node [8].

C. GPU Direct RDMA

NVIDIA GPUDirect [9] enables 3rd-party devices to directly access CUDA device memory, which reduces the overhead of multiple copies to memory. NVIDIA GPUDirect remote direct memory access (RDMA) allows GPU devices to directly exchange data with a remote device across the cluster. It also allows devices on the same PCIe bus to directly transfer data among each other.

III. The Proposed Design

We delve into the details of the hybrid GPU-aware IPC-advanced designs by first describing the intra-node component and then elaborating on the inter-node implementation. We implemented our designs based on MVAPICH2-GDR. In section III-A, we will discuss using IPC-based techniques to overlap intra-node communication with the inter-node data transfer. In section III-B, we will illustrate how to integrate the IPC-based design into the communication path in order to create this overlap in the inter-node environment. We will also detail how we extended our GPU-aware IPC-advanced Alltoall designs to Alltoallv in section III-C.

A. Intra-Node

Traditionally, Alltoall communication within a single node is implemented using multiple send-recv pairs. However, with the utilization of IPC in the intra-node environment, the IPC buffer pointers can be exchanged and utilized by other GPUs to create this intra-node transfer, without utilizing other protocols or the send-recv implementation. We utilize IPC and the heavily connected GPUs systems to apply two techniques: a kernel-based and a memcpy-based implementation:

- Kernel-based IPC: launch a CUDA kernel and pass an IPC buffer pointer to initiate the data exchange across GPUs within a node.
- Memcpy-based IPC: call the CUDA API `cudaMemcpyAsync` and pass an IPC buffer pointer to copy data from device to device within a node.

For both kernel-based and memcpy-based implementations, we use the same algorithm to exchange/copy data. To avoid unbalanced communication within some GPUs in a certain time, we dispatch the workload in a specific order: in the first step, each GPU sends data to itself, and then it starts to send data to the $(rank + i)^{th}$ at the i^{th} step, where $rank$ is the rank of the sender, and i is the iteration. This algorithm spreads the receiving time of each GPU as much as possible. This ensures that at any time, a GPU receives data from only one other GPU. After calling either kernel-based or memcpy-based function we implemented, we call `cudaStreamSynchronize` to make sure data exchanging accomplished.

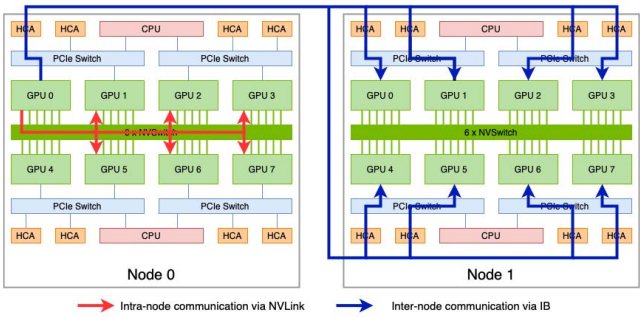


Fig. 5. GPU-aware IPC-advanced Alltoall Design on 2 nodes - 16 GPUs: On Node 0, GPU 0 communicates with all other GPUs within the node using IPC (red arrow) and the inter-node transfer is depicted by the blue arrow.

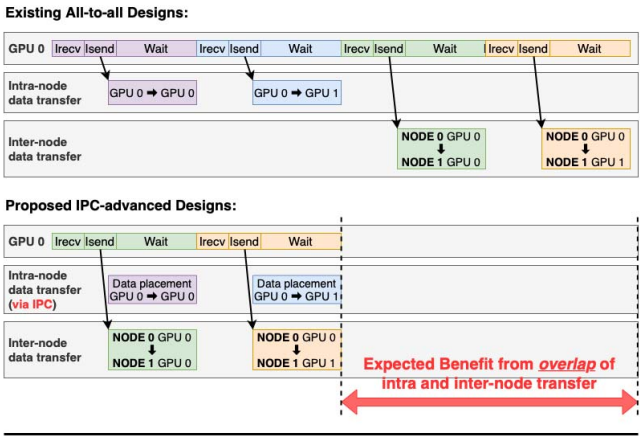


Fig. 6. The proposed IPC-advanced Designs for GPUs provide overlap potential of intra-node and inter-node communication through utilizing zero-copy load store IPC mechanisms. Compared to existing designs that do not have this overlap, the proposed designs are expected to demonstrate enhanced performance. This example depicts a 2 Nodes, 2 GPUs per Node Alltoall scenario.

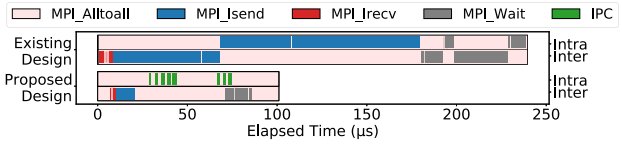


Fig. 7. Timeline comparison of pairwise sendrecv-based (existing) and IPC-advanced (proposed) Alltoall on 2 nodes (ThetaGPU) at 64K.

These two implementations are optimal for different message ranges. Kernel-based IPC has lower overhead for small messages, but the memcpy-based IPC implementation outperforms it as the message size increases. To achieve the lowest latency for a wide range of message sizes, our design takes advantage of both schemes by using a hybrid approach. For example, Figure 4 shows the latency by using different implementations on ThetaGPU within 1 node. We observe that the kernel-based implementation performs well under 256KB, while the memcpy-based implementation has better numbers for message sizes greater than 512KB.

Although IPC-based communication has benefits for large-message transferring within a single node, there are overheads for small messages. Figure 4 shows that our kernel-based and memcpy-based implementation has around 30 µs and 50 µs overhead, respectively for small message sizes. Based on this observation, we introduce the optimized Alltoall design with existing Alltoall algorithms and techniques to deal with small messages. Based on these findings, we design a hybrid algorithm for intra-node communication. We utilize the optimized Alltoall design for small messages, proposed kernel-based IPC for medium messages, and proposed memcpy-based IPC for large messages. The toggle of the implementations can be controlled by $IPC\_threshold$ and $memcpy\_threshold$.

B. Inter-Node

In the inter-node environment, the data transfer path can be categorized into two types: the intra-node and the inter-node communication. Figure 5 shows an example of the communication technique that can be used in a 2-node, 8 ppn (process per node) environment. For intra-node communication, we can utilize our IPC-based algorithm introduced in section III-A directly. For inter-node communication, we have several protocols to deal with different message size ranges. We use NVIDIA GDR Copy library and GDR loopback technique to transfer small messages, and use GPU Direct RDMA to transfer large messages. By calling non-blocking MPI point-to-point calls, `MPI_Isend` and `MPI_Irecv`, we overlap the intra-node and inter-node communications to reduce the latency.

Similar to intra-node, based on the finding depicted in figure 4, we first check whether the $sendcount$ is less than the $IPC\_threshold$, if so, our design falls back to optimized Alltoall communication, or it will perform the IPC-advanced Alltoall. Since inter-node communication usually takes more time, in our implementation, we call non-blocking `MPI_Isend` and `MPI_Irecv` at the beginning, and have `MPI_Wait` before the end of the algorithm to ensure the completion of the point-to-point calls. During the inter-node communication, we call our IPC-based implementation to perform the intra-node communication. This implementation also works in an intra-node environment since in such case, no MPI call is called but the intra-node designs still work well. Figure 6 explains how our proposed IPC-advanced designs provide overlap potential of intra-node and inter-node communication through utilizing zero-copy load store IPC mechanisms.

Figure 7 shows the comparison of the timeline profiling results using our proposed designs and an existing sendrecv-based implementation on 2 DGX-A100 nodes with 16 GPUs (8 GPUs per node). The profiling tools used include TAU and CUDA CUPTI libraries. The timeline demonstrates the details of different MPI calls and the IPC data exchange phase in rank 0. We observe that the existing sendrecv-based implementation spends the most time on `MPI_Isend` and

MPI_Wait. Specifically, there are 16 send-recv pairs and wait calls in the figure. Those pairs cannot be overlapped even though they involve the same amount of intra- and inter-node communication, which dominates the total running time (239.63 µs). On the other hand, there are only 8 send-recv pairs and wait calls in our proposed designs. The other calls for intra-node communication are replaced by our IPC-advanced implementation, which is shown in green in the figure. It saves approximately 50% of the total running time. The saved time is not only coming from the reduction of send/recv calls, but the overlap of the intra-node communication using IPC and the inter-node communication using send/recv calls.

C. Extension to Alltoallv

The most significant challenge for extending Alltoall to Alltoallv designs using IPC-advanced technique is that the send/recv offsets are unknown. In MPI_Alltoallv, there are parameters called sdispls and rdispls. They record the offset from sendbuffer/recvbuffer where the process should send/place data. The information is only related to the current rank. For example, as a sender, the process only knows which piece of data would be sent based on sdispls, but it does not know where the data would be placed after it arrives at another process; as a receiver, it knows where to place the received data according to rdispls, but it would never know which offset the data is coming from. The basic sendrecv-based algorithm works well with this constraint. However, IPC-advanced algorithm is limited by the un-shared information among processes. Either kernel-based or memcpy-based implementation needs the destination offset of the array to assign or copy data directly. To address this challenge, we exchange the destination offset in advance before performing IPC-advanced data transferring. To be specific, we gather the corresponding offsets from rdispls in different processes. There may be some overheads for utilizing gather calls, but due to the benefits from using IPC-advanced design, this overhead is hidden.

Alltoall communication can be considered as a special case of Alltoallv communication, and when MPI_Alltoallv is called, eventually it would require the same behavior as MPI_Alltoall. Given that we are able to optimize Alltoall communication more efficiently (recall that MPI_Alltoall has no gather overhead described in the previous paragraph), our designs will fall back to Alltoall communication once we find that the users expect Alltoall behavior but use Alltoallv calls.

To identify this scenario, our design uses the following optimizations: 1) scan the sendcount (recvcount) array to find out the local maximum and minimum of the sendcount (recvcount) in each rank, 2) use all-reduce calls to get the global maximum and minimum of sendcount (recvcount) among all ranks, then 3) compare the global maximum and minimum. If the values are the same, it indicates that elements in the sendcount (recvcount) array among all ranks are the same. Since the sendcount (recvcount) are the same, the send (recv) offsets can be easily deduced. It is equivalent to the behavior of Alltoall communication, so we can fall back and take advantage of our GPU-aware IPC-advanced Alltoall designs.

IV. Evaluation

A. Experimental Setup

The following evaluations were conducted on the ThetaGPU cluster at Argonne Leadership Computing Facility [10] and on the Lassen cluster at Lawrence Livermore National Laboratory. ThetaGPU is comprised of 24 NVIDIA DGX A100 nodes. Each node is equipped with 2 AMD Rome CPUs, 1TB DDR4 memory, and 8 NVIDIA A100 Tensor Core GPUs. The NVIDIA DGX A100 GPU has 40GB HBM2, and is connected with the second generation NVIDIA NVSwitch. Also, each node is connected with Mellanox ConnectX-6 VPI HDR InfiniBand/Ethernet network adapters and runs with Mellanox OFED 5.1, CUDA driver version 450.142.00, and CUDA version 11.0. The cluster also includes 20 Mellanox QM9700 HDR200 switches wired in a fat-tree topology. Lassen is the #26-ranked machine in the TOP500 [11] as of November 2021 and consists of 792 GPU nodes each with four 16 GB memory NVIDIA Volta V100 GPUs. Each node has two 44-core IBM Power 9 architecture CPUs.

Benchmark-level evaluation: To evaluate the performance of our proposed Alltoall design, we utilize the osu_alltoall test from the OSU Micro-Benchmarks (OMB) [12] suite version 5.8. It reports the latency and bandwidth of point-to-point and collective MPI operations at different message sizes. And for NCCL, we use alltoall from NVIDIA NCCL Tests 2.11.0 as the benchmark. To compare our design, we utilize NCCL 2.11.4, Spectrum-MPI 10.3.1, MVAPICH2-GDR 2.3.6, and OpenMPI 4.1.1 + UCX 1.11.1 as the baselines.

Application-level evaluation: DeepSpeed [13] is a popular distributed DL framework built on top of the PyTorch DL framework. Two of DeepSpeed's key features are memory efficiency for large DL models and efficient parallelism schemes for DL training on large-scale systems. Given these features, DeepSpeed has recently been applied to large-scale DL models that require the Alltoall collective operation. We implemented and applied a communication benchmark built on top of DeepSpeed to measure the throughput of applying Alltoall to PyTorch tensors of varying sizes. Specifically, we use DeepSpeed 0.5.3 with PyTorch 1.9.0, MVAPICH2-GDR 2.3.6, NCCL 2.10.3, and CUDA 10.2.89.

The heFFTe [2] application provides a highly efficient Fast Fourier Transform (FFT) library which supports GPU kernels. In a 3D FFT computation, the workload is divided into 2D or 1D arrays for data exchange. In one iteration, it may require different scales of Alltoallv communication simultaneously within different groups of processes, which makes the communication pattern more complicated. heFFTE also supports using Alltoall communication with data padding.

One limitation we faced in our evaluation was ensuring that all libraries used in the evaluation have support for the datatype used in heFFTe. heFFTe uses MPI_DOUBLE_COMPLEX as the major datatype, which is not supported by NCCL

ncclDataType_t [14]. Additionally, NCCL does not have pre-implemented alltoall or alltoallv APIs, requiring users to implement their own designs by calling ncclSend and ncclRecv. Hence, the users must have knowledge of communication primitives and need to modify or even redesign their programs if they were to utilize NCCL APIs.

The PSDNS mentioned in the following evaluations refers to kernel-based Fourier pseudo-spectral numerical simulation application implemented in [15]. It applies to NVIDIA libraries, such as cuFFT, to accelerate the computation on GPUs. PSDNS utilizes MPI_Alltoall to exchange the transposed data in the 3D-FFT kernel. Currently, the implementation is only supported by the IBM XL compiler. Hence, we only evaluate PSDNS on Lassen.

B. Performance Evaluation

We evaluate the performance of our Alltoall design by running osu_alltoall benchmark in OMB on ThetaGPU from 1 node with a total of 8 GPUs to 16 nodes with a total of 128 GPUs, and on Lassen from 1 node with 4 GPUs per node to 16 nodes with a total of 64 GPUs. On ThetaGPU, we use 4KB as the threshold to switch from optimized Alltoall designs to IPC-advanced designs, and use 512KB as the threshold to switch from kernel-based implementation to memcpy-based implementation. We show the maximum message size of 4MB in each graph, but note that it indicates the message size of each process, Figure 8 shows the latency numbers compared to MVAPICH2-GDR, NCCL and OpenMPI + UCX. To illustrate the details more clearly, we separate the figures by message sizes. Figures 8(a) and 8(d) show the performance on a single node, which indicates that it uses our proposed IPC-based Alltoall design entirely.

For small messages in Figure 8(a), the proposed design achieves a latency of 9.48 µs at 16 bytes message size, approximately 39%, 4.1x and 3.2x better performance compared to MVAPICH2-GDR, NCCL and OpenMPI + UCX. For the medium and large messages, our design also gives better latency numbers against MVAPICH2-GDR, NCCL and OpenMPI + UCX by 1.5x, 12% and 17.4x at 8K message size, and 41%, 14% and 2.5x at 4MB message size. Figure 4 shows the performance comparison between Kernel-based IPC and Memcpy-based IPC designs. It verifies that our IPC-advanced design performs well in any message size on one node. Note that our design uses the optimized Alltoall design only because it has lower latency than the IPC-advanced algorithm for small messages. If we use the IPC-advanced design for message sizes less than 8KB, it can still outperform NCCL and OpenMPI + UCX.

Figures 8(b) and 8(e) show the performance numbers using 32 GPUs on 4 nodes. For large messages, Figure 8(e) shows that the proposed design takes 445.8 µs for 256KB message size, approximately 20% better performance than NCCL. Figures 8(c) and 8(f) show the performance benefits in a larger scale scenario involving 128 GPUs on 16 nodes. It performs similar to the results in Figure 8(b) and 8(e). The proposed design takes 3724.5 µs at 512KB, approximately 44% improved performance over NCCL.

To verify the feasibility of our design on other platforms, we have carried out similar benchmark-level evaluations on Lassen, as indicated in Figure 9. Figure 9(a) shows the performance of small messages. For example, the proposed design takes only 3.3 µs for 16B, approximately 10% better performance than MVAPICH2-GDR. Figure 9(d) shows the proposed design taking 736.3 µs at 4MB, approximately 8% better performance than NCCL. Figures 9(c) and 9(f) show the performance on 16 nodes, 64 GPUs. For this configuration, NCCL has performance degradation for small message sizes. However, if we compare the latency at 4KB, our design takes only 168.7 µs and it's approximately 13% and 10.9x better performance than MVAPICH2-GDR, and NCCL, respectively.

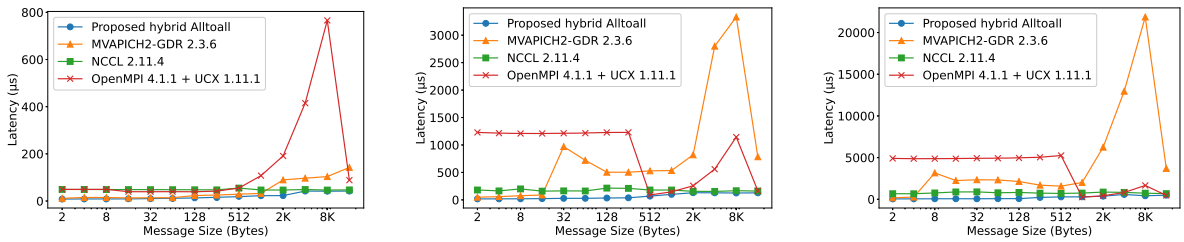
C. Application-Level Evaluation

To evaluate the benefits of our design for real applications, we conducted application-level experiments by running a DeepSpeed communication benchmark for DL applications, and heFFTe and PSDNS for HPC applications.

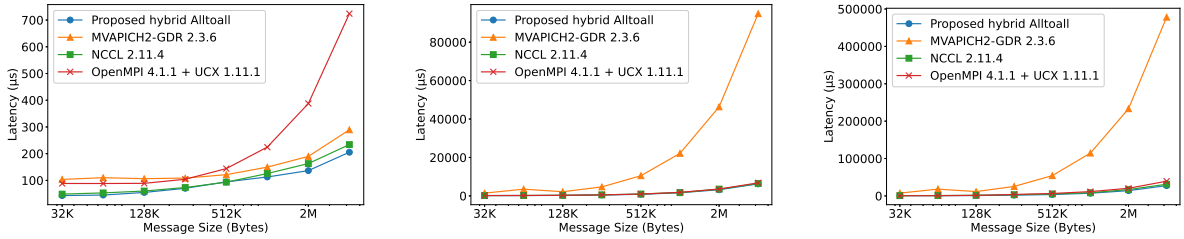
Figure 10 and figure 11 show the evaluation results of DeepSpeed. It measures the throughput of applying Alltoall to PyTorch tensors of varying sizes. Figure 10 shows the throughput per GPU under different tensor sizes compared to MVAPICH2-GDR, NCCL, and OpenMPI + UCX on ThetaGPU. Figure 10(a) shows the results on one node. Our proposed design has better throughput in every Tensor size. For example, we have 7.6 Gbps at 4Kx16 on ThetaGPU; and 1755 Gbps at 16Mx16, which gives 27% better performance than NCCL, on ThetaGPU. In multi-node environment, the throughput shown in Figures 10(b) and 10(c) also outperform against other baselines in every Tensor size. The throughput of our proposed design reaches 192 Gbps at 16Mx16 on 8 nodes, which is 7% NCCL on ThetaGPU.

Figure 11 shows the throughput per GPU under different tensor sizes compared to NCCL on Lassen. Figure 11(a) shows the results on one node. Our proposed design also has better throughput in every Tensor size. For example, we have 26 Gbps at 2Kx4x4, which gives 52% better performance compared to MVAPICH2-GDR on ThetaGPU. In multi-node environment, the throughput shown in Figures 11(b) and 11(c) also outperform against NCCL for every Tensor size. The throughput of our proposed design reaches 1.75 Gbps at 128xx256x4 on 64 nodes, which is 5.7x better than NCCL. Note that MVAPICH2-GDR and Spectrum MPI get segmentation faults above 8 nodes, and OpenMPI + UCX also gets segmentation faults above 64 nodes and are therefore not depicted on these graphs.

Figures 12 and 13 show the throughput of running heFFTe with Alltoall and Alltoallv using different libraries. The throughput numbers are the performance numbers reported by heFFTe itself, which is the overall throughput of the application. Note that NCCL is unavailable here because it doesn't support MPI_DOUBLE_COMPLEX.

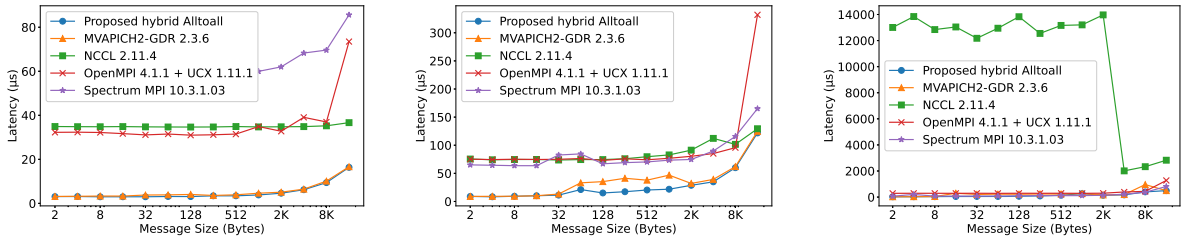


(a) 8 GPUs (1 node 8 ppn) - Small Messages (b) 32 GPUs (4 nodes 8 ppn) - Small Messages (c) 128 GPUs (16 nodes 8 ppn) - Small Messages

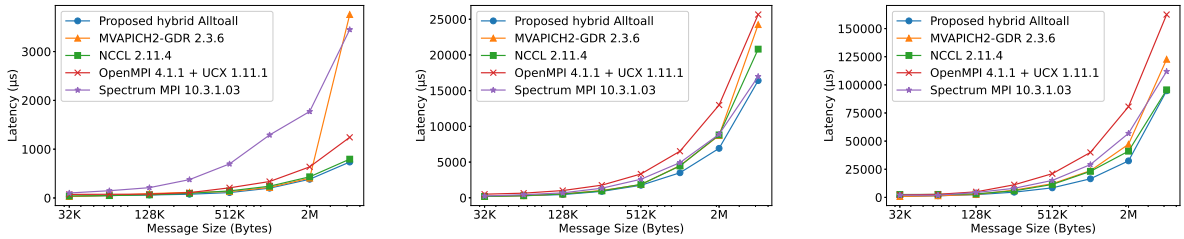


(d) 8 GPUs (1 node 8 ppn) - Large Message (e) 32 GPUs (4 nodes 8 ppn) - Large Message (f) 128 GPUs (16 nodes 8 ppn) - Large Message

Fig. 8. Comparison of Alltoall Operations between Proposed hybrid GPU-aware GPU-aware IPC-advanced and other baselines on ThetaGPU



(a) 4 GPUs (1 node 4 ppn) - Small Messages (b) 16 GPUs (4 nodes 4 ppn) - Small Messages (c) 64 GPUs (16 nodes 4 ppn) - Small Messages

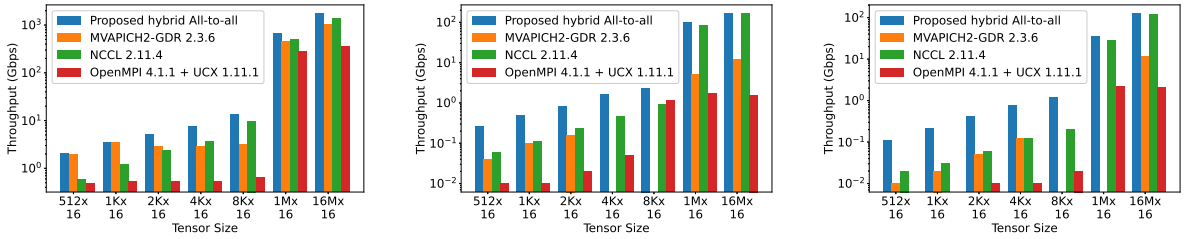


(d) 4 GPUs (1 node 4 ppn) - Large Message (e) 16 GPUs (4 nodes 4 ppn) - Large Message (f) 64 GPUs (16 nodes 4 ppn) - Large Message

Fig. 9. Comparison of Alltoall Operations between Proposed hybrid IPC-advanced and other baselines on Lassen

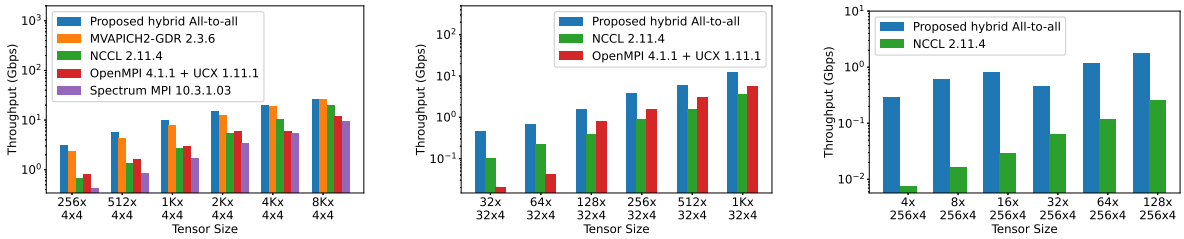
Figures 12(a), 12(b), 12(c) show the results using Alltoall communication on ThetaGPU. For every message size and scale, our proposed designs reach the highest throughput. For example, we obtain 191 GFlops/s at message size $1K^3$ using Alltoall, which gives 1.4x and 14.9x better performance than MVAPICH2-GDR and OpenMPI + UCX on 16 ThetaGPU nodes. Figures 12(d), 12(e), 12(f) show the results using Alltoallv communication. We obtain 4,491 GFlops/s at message size $2K^3$ using Alltoallv, which gives 17.8x and 27x better performance than other baselines on 16 ThetaGPU nodes.

Figure 13 shows similar evaluations for heFFTe on Lassen. The proposed designs give the highest throughput against MVAPICH2-GDR and Spectrum MPI. Note that there are issues in running heFFTe with OpenMPI + UCX on Lassen,



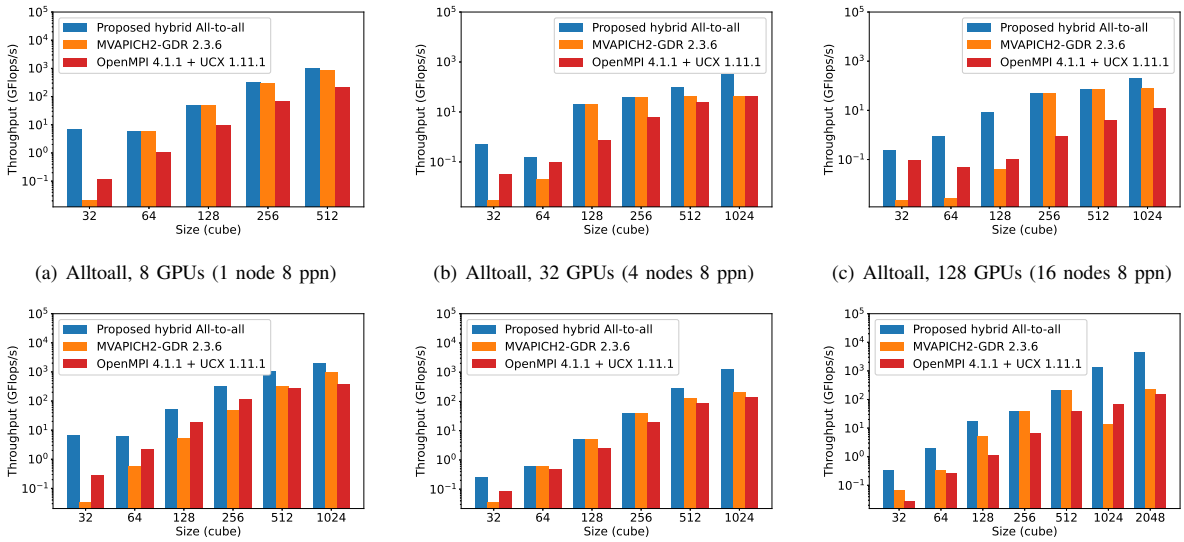
(a) 8 GPUs (1 node 8 ppn) (b) 32 GPUs (4 nodes 8 ppn) (c) 64 GPUs (8 nodes 8 ppn)

Fig. 10. Comparison of application-level (DeepSpeed) Alltoall Operations on ThetaGPU



(a) 4 GPUs (1 node 4 ppn) (b) 32 GPUs (8 nodes 4 ppn) (c) 256 GPUs (64 nodes 4 ppn)

Fig. 11. Comparison of application-level (DeepSpeed) Alltoall Operations on Lassen



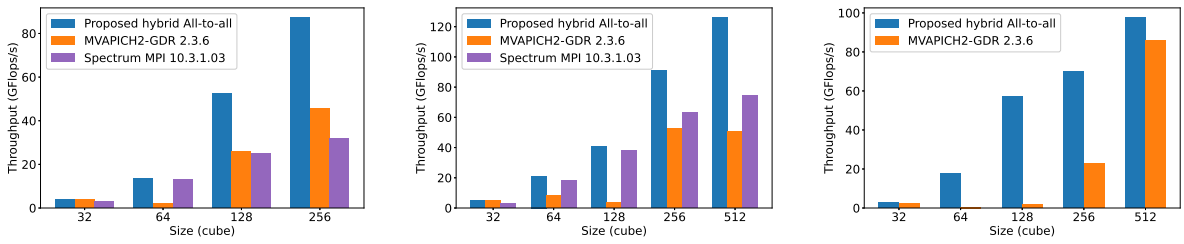
(a) Alltoall, 8 GPUs (1 node 8 ppn) (b) Alltoall, 32 GPUs (4 nodes 8 ppn) (c) Alltoall, 128 GPUs (16 nodes 8 ppn)

(d) Alltoallv, 8 GPUs (1 node 8 ppn) (e) Alltoallv, 32 GPUs (4 nodes 8 ppn) (f) Alltoallv, 128 GPUs (16 nodes 8 ppn)

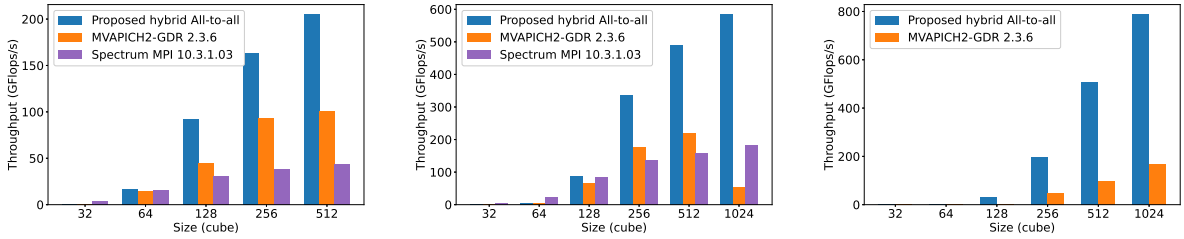
Fig. 12. Comparison of application-level (heFFTe) Alltoall and Alltoallv Operations on ThetaGPU

and we gets errors for running heFFTe with Spectrum MPI above 32 nodes.

We also evaluated our designs with another HPC application, PSDNS, on Lassen. Figure 14 shows the latency of PSDNS from 1 node to 64 nodes. Our proposed designs give the lowest latency for every message size and scale. For example, on one node, our design is 16%-33% faster than MVAPICH2-GDR. On 16 nodes, our design has only 1.05 sec per iteration at size $1K^3$, which is 71% better than MVAPICH2-GDR. Note that PSDNS should be built with IBM XL compiler, and there are issues building OpenMPI + UCX with IBM XL compiler. Also, running PSDNS with Spectrum MPI has errors above two nodes, so we only include MVAPICH2-GDR as the baseline here.

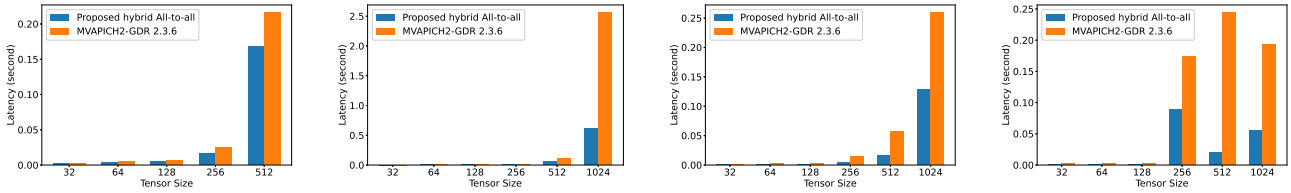


(a) alltoall, 8 GPUs (2 nodes 4 ppn) (b) alltoall, 32 GPUs (8 nodes 4 ppn) (c) alltoall, 128 GPUs (32 nodes 4 ppn)



(d) Alltoallv, 8 GPUs (2 nodes 4 ppn) (e) Alltoallv, 32 GPUs (8 nodes 4 ppn) (f) Alltoallv, 128 GPUs (32 nodes 4 ppn)

Fig. 13. Comparison of application-level (heFFTe) Alltoall and Alltoallv Operations on Lassen



(a) 4 GPUs (1 node 4 ppn) (b) 16 GPUs (4 nodes 4 ppn) (c) 64 GPUs (16 nodes 4 ppn) (d) 256 GPUs (64 nodes 4 ppn)

Fig. 14. Comparison of application-level (PSDNS) Alltoall Operations on Lassen

V. Related Work

While many works have designed MPI_Alltoall algorithms on host buffers [16]–[19], few previous works have explored MPI_Alltoall algorithms specifically for device buffers on GPU-dense systems [20]. Further, most existing GPU MPI_Alltoall algorithms use pipelining [18] [17] or non-blocking methods [21] to hide MPI_Alltoall's relatively high communication overhead. With the advent of the NVIDIA DGX family of HPC systems, the ratio of GPUs to CPUs (number of sockets) on recent systems has increased. Even fewer studies exist on MPI_Alltoallv. Some of the past designs for MPI_Alltoallv focus on CPU systems [22] only.

NVIDIA DGX systems have been a testbed for new efficient collective communications in many prior works. Awan et al. investigated efficient algorithms for MPI_Bcast on DGX systems [23], [24]. The work by Cai et al. [25] demonstrated performance benefits on a DGX system by synthesizing collective algorithms for a specific topology.

Deep learning (DL) applications are increasingly dependent on dense collectives and distributed neural networks. Past works have optimized the MPI_Allreduce collective for distributed DL training [26], yet the MPI_Alltoall collective is increasingly becoming a bottleneck in DL training for models in recommendation [27], and language [28], [29]. Distributed deep learning frameworks have recently accounted for this by adding Alltoall support [13], [30], [31]. Given that recent studies have shown that CUDA IPC is vital to efficient MPI_Allreduce for DL training [32], we expect similar benefits for DL training with MPI_Alltoall.

VI. Conclusion

With the emergence of modern GPU systems equipped with faster, denser, and more connected GPU compute nodes, it is important that communication middleware, such as MPI, utilizes these advancements to provide enhanced performance on these novel systems. In this paper, we improved the performance of GPU-based Alltoall and Alltoallv MPI collective calls on dense GPU systems and proposed a new GPU-aware IPC-advanced design. Considering the different properties of implementations, we have developed a hybrid strategy to use the best communication mechanism to reduce the overhead. For an intra-node environment, we used a kernel-based IPC implementation for smaller messages and a memcpy-based

IPC implementation for larger messages. For an inter-node environment, we utilized the GPU Direct RDMA library to transfer data across nodes and exploited overlap with the intra-node communication by using the proposed IPC-advanced designs. The evaluations have shown that the proposed designs outperform the baseline by 4.1x and 2.5x for small and large messages on one ThetaGPU node, and by 76% for large messages on 16 ThetaGPU nodes. For the DL application DeepSpeed, our proposed designs demonstrate up to 59x better performance on ThetaGPU. In HPC applications heFFTe and PSDNS, our proposed designs reach approximately 27x and 71% better performance on ThetaGPU and Lassen, respectively. As future work, we plan to extend our designs to other common collectives, such as gather and scatter.

VII. Acknowledgement

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

References

[1] A. Ayala, "heFFTe profiler," https://heffte.icl.utk.edu/, 2021.

[2] A. Ayala, S. Tomov, A. Haidar, and J. Dongarra, "heffte: Highly efficient fft for exascale," in *Computational Science – ICCS 2020*, V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Eds. Cham: Springer International Publishing, 2020, pp. 262–275.

[3] H. Wang, S. Potluri, D. Bureddy, C. Rosales, and D. K. Panda, "Gpu-aware mpi on rdma-enabled clusters: Design, implementation and evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2595–2605, Oct 2014.

[4] D. K. Panda, H. Subramoni, C.-H. Chu, and M. Bayatpour, "The mvapich project: Transforming research into high-performance mpi library for hpc community," *Journal of Computational Science*, p. 101208, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877750320305093

[5] Open MPI, "Open MPI: Open Source High Performance Computing," https://www.open-mpi.org/, 2004, Accessed: March 17, 2022.

[6] IBM, "IBM Spectrum MPI: Accelerating high-performance application parallelization," https://www.ibm.com/us-en/marketplace/spectrum-mpi, 2018, Accessed: March 17, 2022.

[7] NVIDIA, "NCCL," https://github.com/NVIDIA/nccl, 2016.

[8] S. Potluri, H. Wang, D. Bureddy, A. K. Singh, C. Rosales, and D. K. Panda, "Optimizing MPI Communication on Multi-GPU Systems Using CUDA Inter-Process Communication," in *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*, May 2012, pp. 1848–1857.

[9] NVIDIA, "NVIDIA GPUDirect," https://developer.nvidia.com/gpudirect.

[10] Argonne National Laboratory, "Theta/ThetaGPU Machine Overview," https://www.alcf.anl.gov/support-center/theta/theta-thetagpu-overview, 2021, Accessed: March 17, 2022.

[11] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer, "TOP 500 Supercomputer Sites," http://www.top500.org, 1993.

[12] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda, "OMB-GPU: A Micro-benchmark Suite for Evaluating MPI Libraries on GPU Clusters," in *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface (EuroMPI)*, 2012, pp. 110–120.

[13] Microsoft, "DeepSpeed," https://www.deepspeed.ai, 2021, Accessed: March 17, 2022.

[14] NVIDIA, "NCCL API - Types," https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/api/types.html, 2021.

[15] K. Ravikumar, D. Appelhans, and P. K. Yeung, "Gpu acceleration of extreme scale pseudo-spectral simulations of turbulence using asynchronism," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '19. New York, NY, USA: Association for Computing Machinery, 2019. [Online]. Available: https://doi.org/10.1145/3295500.3356209

[16] 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, March 2018.

[17] A. K. Singh, S. Potluri, H. Wang, K. Kandalla, S. Sur, and D. K. Panda, "Mpi alltoall personalized exchange on gpgpu clusters: Design alternatives and benefit," in *2011 IEEE International Conference on Cluster Computing*, Sep. 2011, pp. 420–427.

[18] A. K. Singh, "Optimizing all-to-all and allgather communications on gpgpu clusters," Ph.D. dissertation, The Ohio State University, 2012.

[19] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby, "Efficient algorithms for all-to-all communications in multiport message-passing systems," *IEEE Transactions on parallel and distributed systems*, vol. 8, no. 11, pp. 1143–1156, 1997.

[20] 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)*, 2021, pp. 113–122.

[21] A. Venkatesh, K. Hamidouche, H. Subramoni, and D. K. Panda, "Offloaded gpu collectives using core-direct and cuda capabilities on infiniband clusters," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*, Dec 2015, pp. 234–243.

[22] K. Kandalla, H. Subramoni, K. Tomko, D. Pekurovsky, and D. Panda, "A novel functional partitioning approach to design high-performance mpi-3 non-blocking alltoallv collective on multi-core systems," in *2013 42nd International Conference on Parallel Processing*, 2013, pp. 611–620.

[23] A. A. Awan, K. Hamidouche, A. Venkatesh, and D. K. Panda, "Efficient large message broadcast using nccl and cuda-aware mpi for deep learning," in *Proceedings of the 23rd European MPI Users' Group Meeting*, ser. EuroMPI 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 15–22. [Online]. Available: https://doi.org/10.1145/2966884.2966912

[24] A. A. Awan, C. Chu, H. Subramoni, and D. K. Panda, "Optimized broadcast for deep learning workloads on dense-gpu infiniband clusters: MPI or nccl?" *CoRR*, vol. abs/1707.09414, 2017. [Online]. Available: http://arxiv.org/abs/1707.09414

[25] Z. Cai, Z. Liu, S. Maleki, M. Musuvathi, T. Mytkowicz, J. Nelson, and O. Saarikivi, "Synthesizing optimal collective algorithms," *CoRR*, vol. abs/2008.08708, 2020. [Online]. Available: https://arxiv.org/abs/2008.08708

[26] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. Panda, " NV-Group: Link-Efficient Reduction for Distributed Deep Learning on Modern Dense GPU System," in *The 34th ACM International Conference on Supercomputing (ICS-2020)*, 2020.

[27] M. Naumov, D. Mudigere, H. M. Shi, J. Huang, N. Sundaraman, J. Park, X. Wang, U. Gupta, C. Wu, A. G. Azzolini, D. Dzhulgakov, A. Mallevich, I. Cherniavskii, Y. Lu, R. Krishnamoorthi, A. Yu, V. Kondratenko, S. Pereira, X. Chen, W. Chen, V. Rao, B. Jia, L. Xiong, and M. Smelyanskiy, "Deep learning recommendation model for personalization and recommendation systems," *CoRR*, vol. abs/1906.00091, 2019. [Online]. Available: http://arxiv.org/abs/1906.00091

[28] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[29] J. He, J. Qiu, A. Zeng, Z. Yang, J. Zhai, and J. Tang, "Fastmoe: A fast mixture-of-expert training system," 2021.

[30] N. Shazeer, Y. Cheng, N. Parmar, D. Tran, A. Vaswani, P. Koanantakool, P. Hawkins, H. Lee, M. Hong, C. Young, R. Sepassi, and B. A. Hechtman, "Mesh-tensorflow: Deep learning for supercomputers," *CoRR*, vol. abs/1811.02084, 2018. [Online]. Available: http://arxiv.org/abs/1811.02084

[31] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in TensorFlow," *CoRR*, vol. abs/1802.05799, 2018. [Online]. Available: http://arxiv.org/abs/1802.05799

[32] Q. Anthony, L. Xu, H. Subramoni, and D. K. D. Panda, "Scaling single-image super-resolution training on modern hpc clusters: Early experiences," in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2021, pp. 923–932.