Efficient Personalized and Non-Personalized Alltoall Communication for Modern Multi-HCA GPU-Based Clusters

Kaushik Kandadi Suresh, Akshay Paniraja Guptha, Benjamin Michalowicz, Bharath Ramesh,
Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni, Dhabaleswar Panda
Department of Computer Science and Engineering
The Ohio State University Columbus, USA
{kandadisuresh.1, panirajaguptha.1, michalowicz.2, ramesh.113, abduljabbar.1, shafi.16, subramoni.1, panda.2}@osu.edu

***Abstract*—Graphics Processing Units (GPUs) have become ubiquitous in today's supercomputing clusters primarily because of their high compute capability and power efficiency. Message Passing Interface (MPI) is a widely adopted programming model for large-scale GPU-based applications used in such clusters. Modern GPU-based systems have multiple HCAs. Previously, scientists have leveraged multi-HCA systems to accelerate inter-node transfers between CPUs using point-to-point primitives. In this work, we show the need for collective-level, multi-rail aware algorithms using MPI_Allgather as an example. We then propose an efficient multi-rail MPI_Allgather algorithm and extend it to MPI_Alltoall. We analyze the performance of this algorithm using OMB benchmark suite. We demonstrate approximately 30% and 43% improvement in non-personalized and personalized communication benchmarks respectively when compared with the state-of-the-art MPI libraries on 128 GPUs**

***Index Terms*—MPI, DDT, GPU, HCA, Multi-HCA**

I. Introduction

Graphics Processing Units (GPUs) are one of the accelerators that are gaining prominence in modern super-computing systems. This trend is evident from the fact that eight of the top ten systems on the Top500 [11] list are empowered by GPUs (at the time this paper was written). These accelerators enable supercomputers to run massively parallel application workloads from different domains such as scientific computing and Deep-Learning.

In addition to highly efficient links, some modern GPU systems, such as the NVIDIA-DGX, have multiple on-node Host Channel Adapters (HCA) to enhance inter-node communication. The DGX-A100 systems, for example, feature eight HDR (200 Gbps) InfiniBand HCAs for compute traffic and a total of eight A100 GPUs. One can get 600 GBps NVLink bandwidth per GPU, and a total of 4.8 TBps in full utilization of the HCA channels.

Data exchange is essential to parallel applications that run on dense GPU systems, with MPI being one of the widely used programming models that enable communication for applications running on such systems. These applications exchange data between MPI ranks using point-to-point and collective primitives.

For traditional CPU-to-CPU communication, the multi-rail capability has been used to optimize the MPI primitives. Specifically, round-robin and striping based multi-rail designs have been explored to optimize the inter-node data movement [8]. These designs have been implemented at the point-to-point level. However, if multi-rail designs are directly applied to modern GPU-based clusters, MPI collectives may not able to reap the complete benefits of multi-HCA GPU architecture. This is further elaborated in the motivation section (Section I-A).

A lot of past research has been done on optimizing collectives such as Allreduce and Broadcast. However, dense collectives such as personalized All-to-all (MPI_Allgather) and non-personalized All-to-all (MPI_Alltoall), have not been actively researched. MPI_Alltoall operations widely are used in Fast Fourier Transform (FFT) based applications. Allgather operations are gaining traction recently for performing model parallelism for deep neural network training on GPU clusters [1], [9]. Given these advances, it is vital to conduct an in-depth study and optimize the communication of these collectives on current and next-generation dense-GPU systems. Therefore, we propose a multi-HCA aware algorithm for MPI_Allgather. We then extend this to MPI_Alltoall. We adopt a theoretical model for the performance of the proposed algorithm as well as a benchmark evaluation to substantiate our claims.

A. Motivation

We take MPI_Allgather as a candidate collective for our analysis. Hierarchical algorithms are the state-of-the-art algorithms for GPU based allgather. These algorithms perform exchanges at multiple levels. Typically, there are intra-node exchanges at the first level and inter-node exchanges at the second level. These algorithms employ some kind of overlap between different levels in the hierarchy [5].

When these algorithms are employed in a GPU-based multi-HCA system, they need to rely on point-to-point-based multi-

*This research is supported in part by NSF grants #1818253, #1854828, #1931537, #2007991, #2018627, #2112606, and XRAC grant #NCR-130002.
Thanks to ALCF for providing resources for this study.

rail designs to utilize the benefits of multiple HCAs. In a GPU-based system like the DGX-A100, such a point-to-point design may not be completely able to use all the HCAs because GPU-direct RDMA may not be supported for cross-socket exchanges. For instance, in Figure 1, GPU0 may not be able to directly send data to NIC4, NIC5, NIC6, and NIC7. This means that the data has to be staged through the host which will degrade the point-to-point performance. Therefore, it is necessary for the GPU collectives to be able to directly use GPUs to inject data to the closest HCAs.

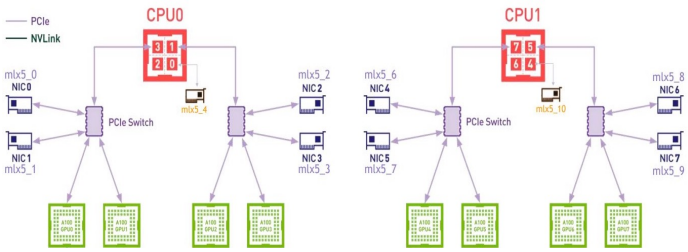


Fig. 1. GPU-HCA topology in DGX-A100 system. [4]

B. Contributions

In this work, we motivate the need to optimize MPI collective algorithms on modern GPU-enabled systems with multiple HCAs by taking All-to-all personalized and non-personalized collectives as a representative communication pattern. We design multi-rail aware approaches for these collectives and evaluate these designs at the micro-benchmark level with the state-of-the-art MPI libraries.

To summarize, this paper makes the following contributions:

1) Motivate the need for a collective level multi-rail design for multi-HCA GPU based systems.
2) Propose a basic and optimized multi-rail design MPI_-Allgather and extend it to MPI_Alltoall.
3) Theoretically analyze the performance of the proposed algorithm by providing performance models for the multi-rail designs.
4) Demonstrate approximately 30% and 43% improvement in non-personalized and personalized communication benchmarks respectively when compared with the state-of-the-art MPI libraries on 128 GPUs .

II. Design and Implementation

In this section, we describe the multi-rail aware collective algorithms for MPI_Allgather and MPI_Alltoall.

A. Basic Framework

Collective algorithms start with each MPI process in a node identifying the closest HCA to the GPU to which the process is mapped. This HCA is then used by each MPI process for all inter-node GPU communications. Each process is mapped to a single HCA in this framework, where it will act as the source of data for its inter-node communication, as opposed to one process acting as a leader and exchanging data on the behalf of other intra-node peers. This way all the HCAs in a node will be utilized.

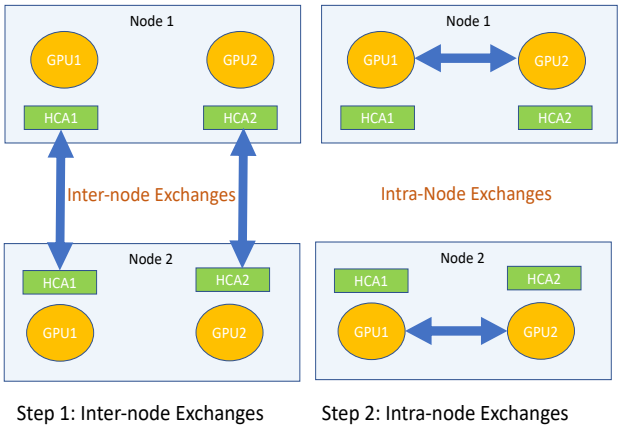


Fig. 2. Steps in the basic multi-rail MPI_Allgather design for a 2 Node, 2 GPU per node MPI_Allgather

B. Proposed Basic Multi-Rail Design for MPI_Allgather

The basic multi-rail design can be divided into inter-node and intra-node part as shown in Figure 2. In the inter-node part, each process first selects its inter-node peer ranks. In Figure 2, GPU1 of nodes 1 and 2 form one inter-node group, and GPU2 of nodes 1 and 2 form another inter-node group. Following this, an inter-node exchange of data happens among all the inter-node groups. This completes the first step of the Multi-Rail design. Now, all the processes in a node will exchange the data obtained from the inter-node phase to complete the Allgather.

Figure 3 shows the flow in the receive buffers after `localCopy` method is called. Each receive buffer initially has its own data marked by its rank number in Figure 3. After the inter-node step, each rank has all the inter-node data for that particular rail. After this, each node has all the data required for MPI_Allgather as each GPU has the corresponding chunk inter-node data. In the next step, a local exchange of data happens which ensures that the GPU receive buffers have the entire set of data that completes the MPI_Allgather.

C. Proposed Optimized Multi-Rail Design for MPI_Allgather

Next we make some observations to optimize the above algorithms. In the above algorithm, one portion of the intra-node exchange can be initiated even before the inter-node phase for which the data source is locally present on the same node. For instance, in Figure 3, in node1, GPU1 and GPU2 can exchange their local data, 0 and 1 independent of the inter-node exchange step. The next observation we make is that since there is a data-dependency in the inter and intra node steps, we can break it into multiple steps to achieve overlap between inter and intra node exchanges.

This enhanced version is shown in algorithm 1. The main `allgatherOpt` function uses the following functions to achieve the overlap. First is `localCopyInit` function which initiates a local copy from send to receive buffer. Then, the `localIntraExchangeInit` function initiates the intra-node exchange that was discussed earlier. The

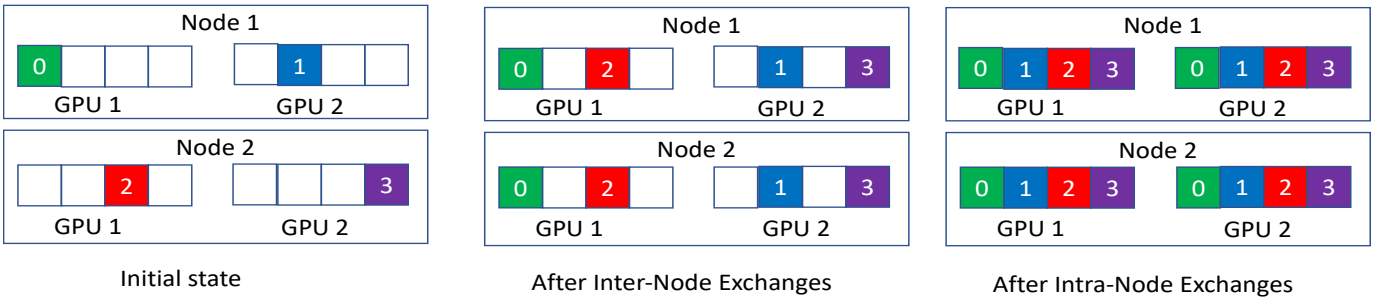


Fig. 3. Receive Buffer State after each step of basic multi-rail allgather algorithm

Algorithm 1: Proposed Optimized Multi-Rail Allgather Algorithm

Input : sb — Send Buffer, rb — Receive Buffer, sz — Size, GPN — GPUs per Node, $numRanks$ — Total processes
Output: rb — Receive Buffer

```
// Main Allgather Function
Function allgatherOpt(sb, rb, sz, GPN, numRanks):
    ns ← Total Number of Steps;
    stepSize ← sz/ns ;
    localCopyInit(sb, rb, sz);
    localIntraExchangeInit(sb, rb, sz, GPN, numRanks);
    interExchangeInit(sb, rb, stepSize, sz, GPN, numRanks, 0);
    interExchangeComplete(0);
    step ← 1 ;
    for step ← 1 to ns − 1 do
        intraExchangeInit(sb, rb, stepSize, sz, GPN, numRanks, step − 1);
        interExchangeInit(sb, rb, stepSize, sz, GPN, numRanks, step);
        intraExchangeComplete(step − 1);
        interExchangeComplete(step);
    intraExchangeInit(sb, rb, stepSize, sz, GPN, numRanks, ns − 1);
    intraExchangeComplete(ns − 1);
```

`interExchangeInit` and `intraExchangeInit` functions each take two additional parameters — step size(sz) and step number($step$). These functions initiate the respective operations for a particular chunk of the data that is transferred for a given step. Each of these init methods has a corresponding complete method that waits for the operations to complete. The `allgatherOpt` function divides the message into ns chunks. Then using `localCopyInit` the intra-node exchange is initiated. Then the inter-node exchange for the first chunk is completed. After this, for $ns-1$ times, the intra-node exchange of chunk $step-1$ is overlapped with inter node exchange of chunk $step$. Finally, the intra-node exchange of each chunk is completed for the last chunk.

D. Performance Modeling of the Multi-Rail Design

In the equations that follow, T_{ns}, T_{ng} refer to the startup time for a single network, GPU transfers respectively. Similarly, B_n, B_g refer to the Bandwidth of the network, GPU inter-connect respectively. N refers to the total node count and GPN refers to the number of GPUs in a Node. The total time for the basic multi-rail scheme can be modeled as the sum of inter and intra-node as shown in equation 1.

$$T_{mrail} = T_{inter} + T_{intra} \tag{1}$$

In the inter-node phase, GPN groups of inter-node exchanges happen at the same time. The cost depends on the type of exchange algorithm used. The inter-node cost using the direct algorithm is:

$$T_{inter} = N \times (T_{ns} + \frac{M}{B_n}) \tag{2}$$

In the intra-node phase, we use a direct exchange method to take advantage of the NVLINK interconnects between GPUs. In this scheme, each GPU exchanges $N \times M$ bytes of data. The total cost for this phase is :

$$T_{intra} = N \times GPN \times (T_{gs} + \frac{M}{B_g}) \tag{3}$$

Thus, the total cost is :

$$T_{mrail} = N \times (T_{ns} + \frac{M}{B_n}) + N \times GPN \times (T_{gs} + \frac{M}{B_g}) \tag{4}$$

In the multi-rail optimized scheme, we split the message into s segments of size m bytes each (which is $\frac{M}{s}$). We first calculate the cost of doing local intra-node exchange, single inter-node, and intra-node segment exchange.

The cost of a local intra-node exchange is :

$$T_{intra-local} = GPN \times (T_{gs} + \frac{M}{B_g}) \tag{5}$$

The cost of a single inter-node segment exchange is :

$$T_{inter-single} = N \times (T_{ns} + \frac{m}{B_n}) \tag{6}$$

The cost of a single intra-node segment exchange is :

$$T_{intra-single} = (N-1) \times GPN \times (T_{gs} + \frac{m}{B_g}) \tag{7}$$

In the above equation, we note that the multiplier is $N-1$ unlike Equation 3, because we initially do the local-intra-node exchange for the entire message size. Now that we have $T_{intra-single}$, $T_{inter-single}$, and $T_{intra-single}$, the final cost for the multi-rail optimized scheme is :

$$T_{mrail-opt} = max(T_{intra-local}, T_{inter-single}) + (s-1) \times max(T_{inter-single}, T_{intra-single}) + T_{intra-single} \quad (8)$$

E. Extension of the Optimized Multi-Rail Design for MPI_Alltoall

We extend algorithm 1 to support MPI_Alltoall. First, the send buffer size becomes $N \times GPN \times sz$ it has personalized data for each remote process. In each of the exchange functions, the step size $StepSize$ becomes $\frac{(N \times GPN \times sz)}{ns}$. Each of these chunks of the send buffer will contain the data to be sent to many processes within the same remote node. Therefore in the `interExchangeInit` each process receives the combined data in a stage buffer from a different node. In the `intraExchangeInit` phase each process scatters the data to multiple processes based on the offsets. The only difference here is each process gets a different set of data. Therefore, the amount of data exchanged in personalized All-to-all is $N \times GPN$ times more than in the non-personalized version.

III. Performance Evaluation

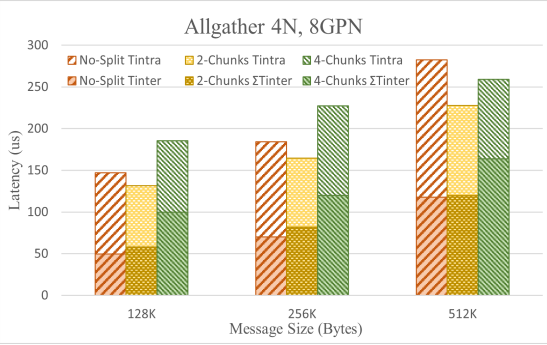


Fig. 4. Performance of Optimized Multi-Rail Scheme for different number of segments

We implement the proposed multi-rail schemes in a GPU-aware MPI library. In this section, we analyze the performance of the proposed multi-rail schemes using MPI_Allgather benchmark from the OSU Micro-Benchmarks (OMB) suite [10]. We also compare the performance of our multi-rail scheme with the performance of other CUDA-Aware MPI libraries such as OpenMPI 4.1.3 with UCX 1.12.1, MVAPICH2-GDR V2.3.7, and NCCL 2.12.10-1. For all our experiments, we report an average of 100 iterations, excluding the 10 warm-up iterations, in three consecutive back-to-back trials per experiment.

A. Experimental Platforms and Setup

We used the ThetaGPU cluster for all our experiments. The ThetaGPU cluster, deployed at the Argonne Leadership Computing Facility (ALCF), contains 24 DGX-A100 nodes with AMD-EPYC processors. The NVIDIA DGX A100 GPU has 40GB HBM2. The GPUs are connected with the third generation NVIDIA NVLink and the second generation NVIDIA NVSwitch. The detailed hardware specifications of these clusters are shown in Table I.

TABLE I
Hardware specification of our test-bed cluster

Specification	ThetaGPU
Processor Family	AMD EPYC
Processor Model	EPYC 7742
Clock Speed	3.4 GHz
Sockets	2
Cores Per socket	64
NUMA nodes	8
CCX Per NUMA	4
RAM (DDR4)	1 TB
Interconnect	IB-HDR(200G) - 8 HCAs
GPU Processor	NVIDIA A100×8
GPU Memory	40GB
Interconnects between GPUs	NVLink-3 and NVSwitch
NVIDIA Driver Version	470.82.01

B. Impact of number of segments

First, we look at the performance of Multi-Rail scheme with OMB Allgather (see section II-D for details). Figure 4 shows the OSU_Allgather latency for different number of chunks on four nodes having eight GPUs per node. We observe that 2 is the optimal number of segments and the performance degrades beyond this. This is attributed to the fact that the total inter-node cost ($T_{inter-single} \times s$) dominates the intra-node cost based on our observation. This also means that the inter-node cost increases as we increase the number of chunks. As explained before, once the increase in this cost outweighs the benefits obtained due to intra-node overlap, the performance starts degrading. This is the trade-off of splitting the messages into multiple chunks. In the subsequent section, we use "Proposed" to denote the optimized multi-rail scheme with the optimal chunk size of two (2).

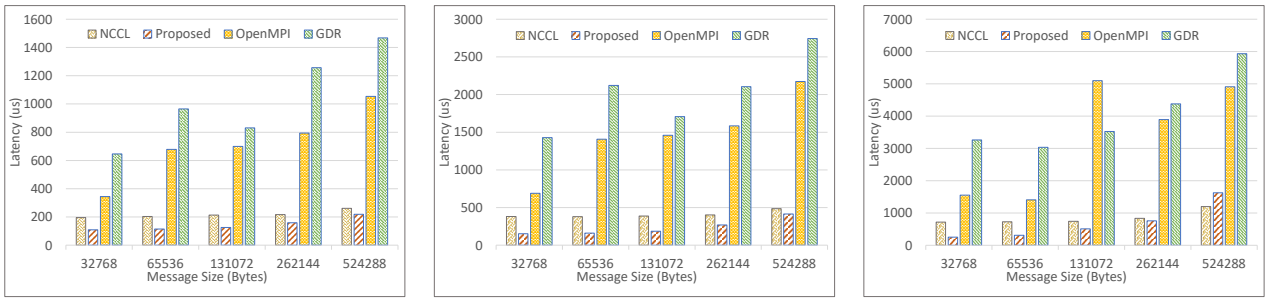
C. Performance of the Proposed compared to other MPI libraries

Figure 5 shows the performance of the proposed multi-rail MPI_Allgather scheme compared to other MPI libraries on 4, 8 and 16 nodes. On four nodes we observe that the proposed scheme performs up-to 42% better than NCCL and at least 2X better than OpenMPI+UCX and MVAPICH2-GDR. On eight nodes, we observe that the proposed scheme performs up-to 60% better than NCCL and at least 3X better than OpenMPI+UCX and MVAPICH2-GDR. On 16 nodes having 128 GPUs, the proposed scheme performs up to 2X better than NCCL.

Figure 6 shows the comparison the propose multi-rail algorithm with MPI_Alltoall on 4, 8 and 16 nodes. We observe that on 128 GPUs, the proposed scheme performs up-to 85% better than OpenMPI+UCX and 88% better than MVAPICH2-GDR.

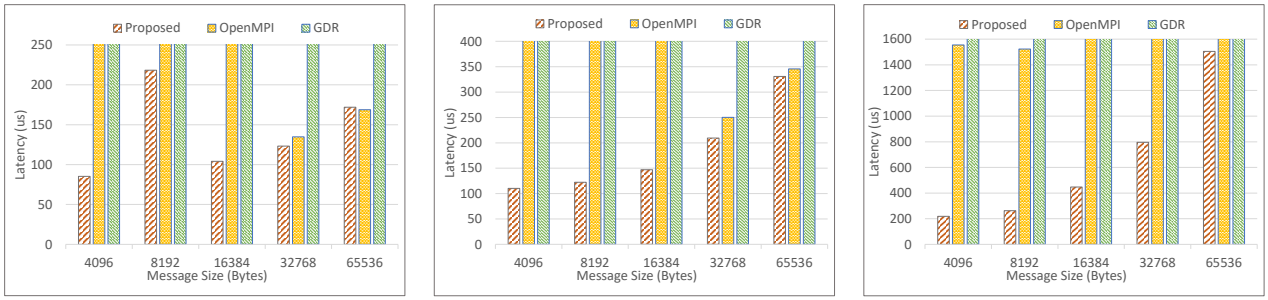
IV. Related Work

Several researchers have targeted dense GPU systems featuring multiple HCAs, such as those enhancing the reduction



(a) 4N, 32GPUs (b) 8N, 64GPUs (c) 16N, 128GPUs

Fig. 5. Performance comparison of MPI_Allgather on ThetaGPU across MPI libraries



(a) 4N, 32GPUs (b) 8N, 64GPUs (c) 16N, 128GPUs

Fig. 6. Performance comparison of MPI_Alltoall on ThetaGPU across MPI libraries

and broadcast steps of PageRank [3], Alltoall communication for FFTs [13], and Allreduce for Deep Learning workloads [12]. However, the approaches adopted by these works were compute-centric and did not consider extending the MPI collectives to exploit multiple HCAs.

Other work includes topology-aware [2], [6] and memory-aware collective algorithms [7], though these approaches are limited by the hardware of the time and may require upgrades as new hardware and topologies emerge.

V. Conclusion

Modern Supercomputers are increasingly using GPU-based compute nodes to accelerate the performance of parallel MPI-based applications. In this work, we used a multi-HCA-based GPU cluster to motivate the need for multi-rail aware algorithms at the collective level. We used MPI_Allgather, MPI_Alltoall as a use case to propose basic and optimized multi-rail aware algorithms. We performed a theoretical and empirical analysis of the proposed algorithm by modeling it and evaluating the performance using different MPI libraries. The proposed scheme shows approximately 30% and 43% improvement in non-personalized and personalized communication benchmark respectively when compared with the state-of-the-art MPI libraries.

References

[1] T. Ben-Nun and T. Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Comput. Surv.*, 52(4), aug 2019.

[2] I. Faraji and A. Afsahi. GPU-Aware Intranode MPI_Allreduce. In *Proceedings of the 21st European MPI Users' Group Meeting*, EuroMPI/ASIA '14, pages 45:45–45:50, New York, NY, USA, 2014. ACM.

[3] S. Kang, A. Fender, J. Eaton, and B. Rees. Computing pagerank scores of web crawl data using dgx a100 clusters. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–4, 2020.

[4] P. Kennedy. HC32 NVIDIA DGX A100 SuperPOD DGX A100 System Topology. https://www.servethehome.com/google-tpuv3-discussed-at-hot-chips-32, Aug 2020.

[5] 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)*, pages 113–122, 2021.

[6] R. Kobus, D. Jüunger, C. Hundt, and B. Schmidt. Gossip: Efficient communication primitives for multi-gpu systems. In *Proceedings of the 48th International Conference on Parallel Processing (ICPP)*. ACM, 2019.

[7] 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*, 29:pp. 542–555, 2018.

[8] J. Liu, A. Vishnu, and D. K. Panda. Building Multirail InfiniBand Clusters: MPI-Level Design and Performance Evaluation. In *Super-Computing Conference*, 2004.

[9] M. Naumov, J. Kim, D. Mudigere, S. Sridharan, X. Wang, W. Zhao, S. Yilmaz, C. Kim, H. Yuen, M. Ozdal, K. Nair, I. Gao, B.-Y. Su, J. Yang, and M. Smelyanskiy. Deep learning training in facebook data centers: Design of scale-up and scale-out systems, 2020.

[10] OSU Micro-benchmarks. http://mvapich.cse.ohio-state.edu/benchmarks/.

[11] E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer. Top 500 supercomputer statistics. http://www.top500.org/statistics/list/.

[12] Y. H. Temuçin, A. H. Sojoodi, P. Alizadeh, B. Kitor, and A. Afsahi. Accelerating deep learning using interconnect-aware ucx communication for mpi collectives. *IEEE Micro*, 42(2):68–76, 2022.

[13] B. Zhou and L. Lu. An effective 3-d fast fourier transform framework for multi-gpu accelerated distributed-memory systems. *The Journal of Supercomputing*, pages 1–19, 2022.