Collaborative Bandwidth-Efficient Intra-Node Allreduce

Amirhossein Sojoodi, Ali Farazdaghi, Hamed Sharifian, Ryan E. Grant, and Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University
Kingston, Ontario, Canada
{amir.sojoodi, m.farazdaghi, hamed.sharifian, ryan.grant, ahmad.afsahi}@queensu.ca

***Abstract*—The concept of GPU-aware multi-path communication and heterogeneous computing is pivotal in enhancing HPC cluster performance. Heterogeneous computing utilizes various computing units to boost distributed applications, while multi-path communication frameworks optimize data transfer and reduce latency. This study proposes a collaborative *Allreduce* collective communication framework that utilizes heterogeneous computing and multi-path communication for multi-GPU systems. Leveraging GPU parallelism and high memory bandwidth, our approach optimizes both data transfer and computing power of the host system. We provide a proof-of-concept implementation and evaluation on a single-node multi-GPU system, comparing its performance to NCCL, UCC, and other algorithms. Our results show that our framework can improve *Allreduce* performance for large message sizes by up to 1.45x compared to NCCL. Our approach can be extended to various topologies and systems, with more extensive evaluations planned for the future.**

***Index Terms*—GPU, Multi-Path Communication, Heterogeneous Computing, Allreduce, Collective Communication**

I. Introduction

In the fast-paced world of High-Performance Computing (HPC), achieving efficient utilization of computational and communication resources remains a critical challenge. The growing demand for processing vast datasets and running complex simulations necessitates innovations that can deliver higher performance and scalability. The two prominent paradigms, heterogeneous computing and multi-path communication, have emerged as key contributors to addressing these demands [10], [18].

Heterogeneous computing enables developers to utilize a variety of computing or communication units at the same time. As demonstrated in previous studies, utilizing computation or communication heterogeneity, can lead to significant performance improvements in HPC applications [2], [15], [9]. For instance, in training LLMs, when the model size exceeds the memory capacity of the Graphics Processing Unit (GPU)s, model states can be spilled to the host memory in a pipeline fashion to remedy the memory constraints [3], [8]. However, leveraging this potential requires careful analysis of the underlying hardware architecture and the applications' requirements. Moreover, with multi-path communication, developers optimize the data movement by utilizing available communication paths. By balancing traffic and reducing congestion, this approach can significantly enhance bandwidth utilization and minimize communication latency. Previous studies have shown that multi-path communication even for single messages can improve the performance of HPC applications when applied with explicit consideration of the underlying hardware architecture [10], [11], [12].

Collective communication operations, such as *Allreduce*, are fundamental building blocks in many HPC applications, including machine learning, scientific simulations, and numerical computations. These operations often dominate the overall execution time, especially in GPU-heavy workloads where data exchange between multiple GPUs can become a bottleneck [1], [13]. As such, optimizing collective communication for multi-GPU systems is a critical area of research. Exploring heterogeneity in the context of collective communication has been shown to improve the performance of these operations [17]. However, existing state-of-the-art solutions, such as NVIDIA Collective Communications Library (NCCL) [5], Unified Communication Collectives (UCC) [14], and Open MPI [6], do not exploit the potential of heterogeneous computing and multi-path communication. These libraries are designed to work with homogeneous systems and do not consider the potential benefits of utilizing both CPU and GPU resources for collective communication.

In this paper, we investigate the integration of GPU-aware multi-path communication and heterogeneous computing to enhance the performance of *Allreduce* collective operations in multi-GPU systems. We propose a collaborative framework that leverages the high memory bandwidth and parallelism of GPUs alongside the computational and interconnect resources of the host system. Our approach is designed to address the challenges in existing state-of-the-art solutions, such as NCCL, UCC, and Message Passing Interface (MPI) with Unified Communication X (UCX). We introduce a proof-of-concept design and implementation of a novel *Allreduce* framework on a multi-GPU node and evaluate its performance against state-of-the-art libraries and several algorithms. Our results demonstrate that our proposed framework can achieve up to 1.45x speedup in *Allreduce* compared to NCCL for very large messages, highlighting its potential for further optimizations and extensions.

The remainder of this paper is organized as follows. Section II describes the design, implementation, and optimizations of our heterogeneous multi-path collective framework. Section III presents the experimental setup, performance evaluation,

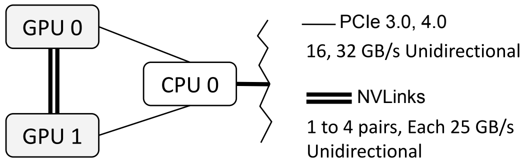

GPU 0
GPU 1
CPU 0
PCIe 3.0, 4.0
16, 32 GB/s Unidirectional
NVLinks
1 to 4 pairs, Each 25 GB/s Unidirectional

Fig. 1: Target topologies for multi-path *Allreduce* framework that utilizes NVLink (any number of pairs) and PCIe paths between two GPUs of the system.

and analysis of the results. Finally, Section IV concludes the paper, summarizing the contributions and discussing future directions.

II. Design and Implementation

Figure 1 illustrates the target topology for our multi-path *Allreduce* framework. In this topology two GPUs are directly connected by NVLink as well as Peripheral Component Interconnect Express (PCIe) interconnect to the system. In such topologies, and depending on the applications, CPU cores are usually used for data staging and control operations, while the GPUs are used for the actual computation. As the application data resides on the GPUs, and the GPUs are directly connected, employing the CPUs to participate in the *Allreduce* might seem counter-intuitive. However, when CPU cores are idle during a GPU collective operation, they could be utilized to accelerate the reduction process, when the data is large enough to saturate the NVLink.

To design and implement our proof-of-concept multi-path heterogeneous *Allreduce* framework, we have considered the following key design goals:

- **Multi-path Communication:** Utilizing all available bandwidth to decrease data transfer overhead between the GPUs.
- **Leveraging Idle CPUs in the Data Path:** Concurrently utilizing the computational capabilities of both CPUs and GPUs to enhance collective communication.
- **Low Overhead:** Implementing a low-overhead pipelined communication scheme to overlap computation and communication tasks along both NVLink and PCIe paths.
- **Data Integrity:** Ensuring data integrity and consistency during data transfers and computations.
- **Reducing CPU Involvement in the Control Path:** Utilizing asynchronous communication primitives to offload communication and computation tasks from the CPUs to the GPUs asynchronously.

A. Framework Architecture

We use *Pairwise Exchange* as the base algorithm for our *Allreduce* design, due to its efficiency for small number of GPUs, and its extensibility to seamlessly support multi-path communication. Moreover, as will be further discussed in Section III, we have not observed significant performance differences between *Pairwise Exchange* and other algorithms for two GPUs. However, for larger number of GPUs, other algorithms such as *Segmented Ring* or *Reduce-Scatter Allgather (RSA)* might be more bandwidth-efficient.

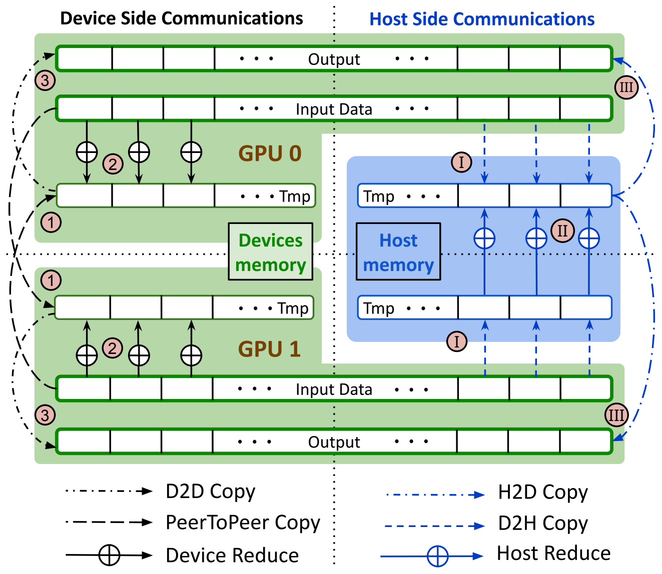


Fig. 2: Multi-path Heterogeneous *Pairwise Exchange Allreduce* on two GPUs using NVLink and PCIe.

Fig. 2 provides a simplified overview of our approach, illustrating how the multi-path heterogeneous *Pairwise Exchange Allreduce* works on two GPUs. The steps on each side of the figure, device (left) and host (right), represent the algorithm tasks that should occur with the specified order and may be concurrent to those of the other side. Our main goal is to maximize concurrency and overlap between the two sides to minimize latency. Therefore, data should be carefully divided into two parts at a certain threshold to be handled by each side. This threshold is determined based on the data size, the available bandwidth of the NVLink and PCIe paths, and the computational power of the GPUs and the host system. For brevity in this work, we have statically tuned this threshold for various message sizes and different systems, and we report the best results.

Additionally, we have implemented our proposed algorithm outside the runtime library using CUDA Driver Application Programming Interface (API), controlling the GPUs via the CUDA *contexts*. We have also utilized CUDA *streams* and CUDA *events* to achieve the desired concurrency and overlap between several simultaneous series of tasks on both sides.

B. *Multi-path Heterogeneous* Allreduce

On the devices side (left), in **Step 1**, each GPU *context* initiates a series of Peer-to-Peer transfers to retrieve the peer data, chunk by chunk, and on a specific *stream*. In **Step 2**, to apply computation on the received chunk with the local data, a computation kernel is enqueued on the corresponding *stream*. Finally, in **Step 3**, the reduced data is stored in the output buffer. For further improvement, this step is packed into the computation kernel to store the results directly into the output buffer from the temporary chunk.

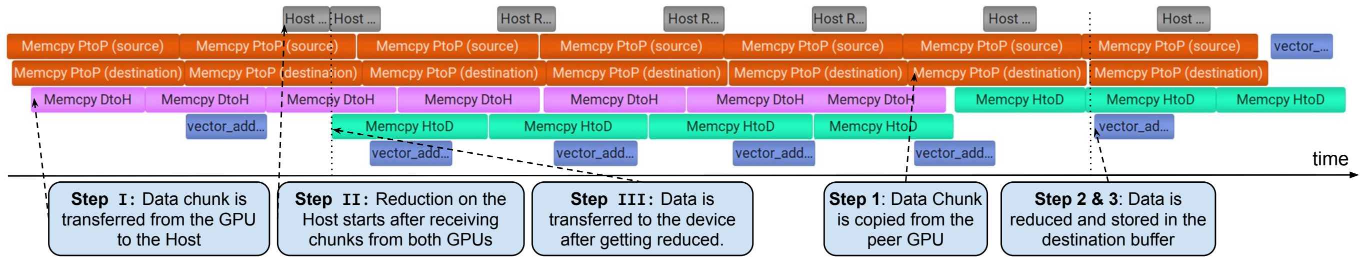


Fig. 3: A summarized NVIDIA Nsight Systems timeline profile of the multi-path heterogeneous *Allreduce* (128 MB) on one of the GPUs on Daisy cluster. Grey and blue boxes are reduction operations on the host and the device, respectively.

The algorithm on the host side (right) performs differently. Instead of the *get* operations, a series of Device-to-Host (D2H) *put* operations are initiated in **Step I** from each CUDA *context* to store their input data to a pre-allocated pinned memory location, chunk by chunk. Then, one of the GPU *contexts* (e.g., GPU 0) synchronizes its specific *streams* with the completion of the D2H *put* operations from both *contexts* through CUDA *events*. After that, for each pair of received chunks from both sides, a *callback* is enqueued into the corresponding *streams* with the `cudaHostFunction` type. When the data becomes ready, in **Step II**, the host function is called to reduce the data and store the results in the output buffer. For this step, we have utilized OpenMP threads to perform the reduction on the host side. Finally, in **Step III**, the reduced data is sent back to the devices through a series of Host-to-Device (H2D) *get* operations by each CUDA *context*.

Figure 3 presents a summarized profile of our multi-path heterogeneous *Allreduce* on one of the GPUs (the other GPU profile is very similar, therefore omitted due to space constraints). The profile clearly illustrates how all interconnects are utilized, and how both CPU and GPU compute resources are leveraged for reduction.

Note that all these operations are initiated by a CPU core in an asynchronous manner, and the GPU *contexts* are synchronized with the completion of data transfers and computations using CUDA *events*. Additionally, special cases such as the last chunk of data or data sizes that are not multiples of the chunk size are carefully handled to ensure data integrity.

III. Evaluation

To evaluate the performance of our framework, we compared it against NCCL (v2.24.3-1), MPI (v5.0.2) configured with UCC (v1.3), UCX (v1.17), and CUDA (v12.6), as well as two commonly used *Allreduce* algorithms: *Pairwise Exchange* and *Segmented Ring*, both implemented outside the runtime libraries, including their pipelined and kernel-only versions. In the pipelined versions of these algorithms, the data is split into multiple chunks to overlap the algorithm steps, while in the kernel-only versions, both computation and communication tasks are packed into a single kernel, similar to how NCCL performs. These implementations utilize neither multi-path nor heterogeneous techniques, thereby ensuring a fair comparison between our framework and NCCL, by highlighting the sources of performance improvements.

A. Experimental Setup

We conducted a series of experiments on three different GPU configurations from these clusters:

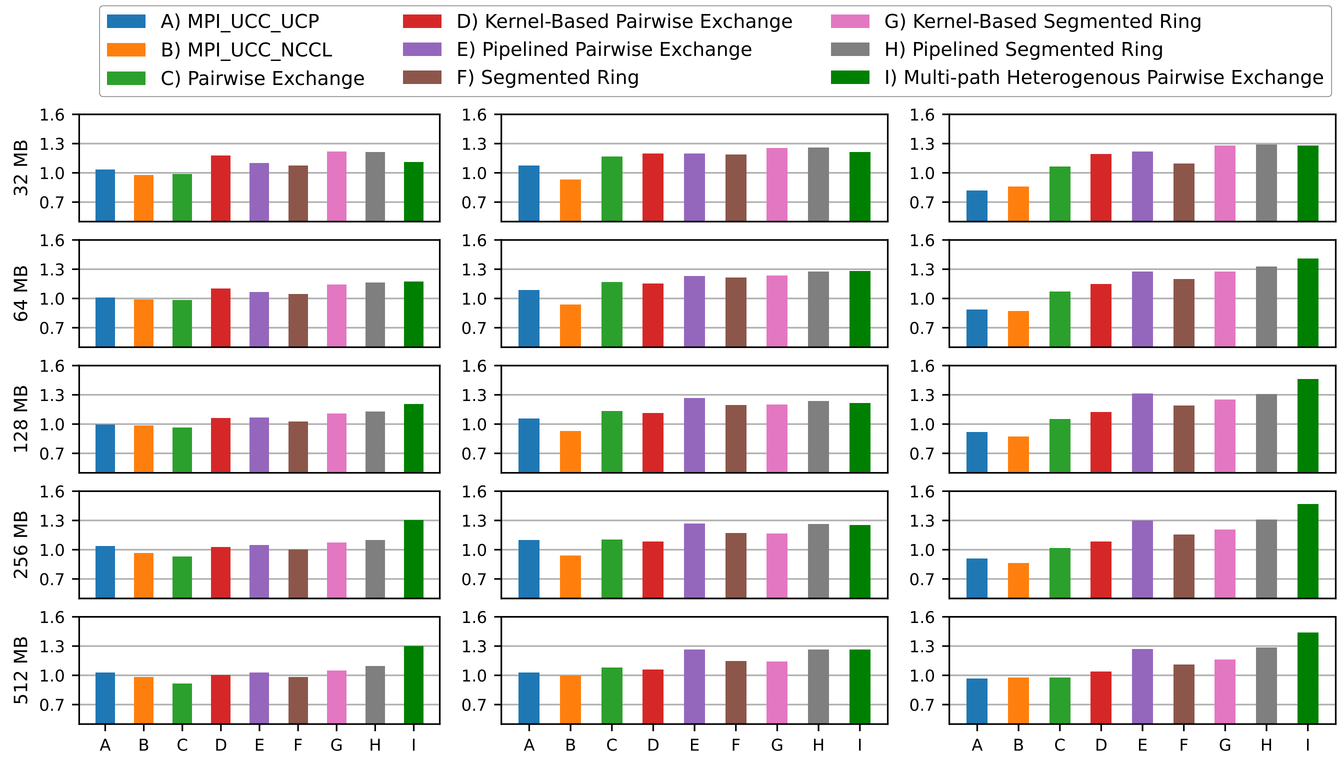
- **Daisy**: A two-socket local node equipped with two NVIDIA A30s on each socket. Each GPU pair is connected by four pairs of NVLink, and each socket has an Intel Xeon Gold 6338, with 32 cores and two threads per core. The two directly connected GPUs and one of the CPUs are configured as a single Non-Uniform Memory Access (NUMA) node.
- **Mist**: Similar to Daisy, but equipped with NVIDIA V100 GPUs, with three pairs of NVLink in-between. The CPUs are Power9 8335-GTH with 16 cores and four HW threads.
- **Narval**: An eight-NUMA node equipped with four NVIDIA A100 GPUs, with a full mesh topology and four pairs of NVLink between any two GPUs. The CPUs are two AMD EPYC 7413 with total of 24 cores with hyper-threading disabled. On this system, each GPU is connected to a single NUMA node, which consists a single memory channel and only 6 cores of one of the CPUs, making it an example that is not ideal for our framework due to the NUMA configuration and the low number of cores available for computation per each NUMA.

We assessed the performance of our framework using two GPUs in each node in these clusters, resembling a two-GPU topology.

B. Performance Analysis

Figure 4 illustrates the speedup of our proposed methodology, (I) *multi-path heterogeneous Pairwise Exchange*, as well as MPI and UCC, against NCCL on Daisy, Narval, and Mist nodes, for message sizes ranging from 32 MB to 512 MB. Evaluating the performance of various algorithms/solutions, we can observe that our proposed approach outperforms NCCL by up to 1.45x for very large messages on Daisy. Other observations include:

- **Observation 1**: By implementing and comparing similar *Allreduce* algorithms to NCCL and UCC (pipelined and kernel-based *Segmented Ring* and *Pairwise Exchange*), we can observe that the runtime libraries (NCCL and MPI) have a small but noticeable overhead compared to their standalone versions.
- **Observation 2**: Our proposed method outperforms all the other implemented algorithms, confirming that utilizing



(a) Speedup on Mist (b) Speedup on Narval (c) Speedup on Daisy

Fig. 4: Speedup of various Allreduce algorithms (including MPI_UCC_UCP, MPI_UCC_NCCL) against standalone NCCL on Daisy, Narval, and Mist nodes, using two GPUs.

multiple paths as well as CPUs and GPUs for computation improves the performance of *Allreduce* for large messages, independent of the communication libraries overhead.

- **Observation 3**: Comparing the improvements from different nodes, we can observe that the performance gain is less significant on Narval. This is due to the NUMA configuration of this system, the low number of CPU cores per NUMA for computation, and the greater performance difference between the NVLink and PCIe.
- **Observation 4**: Pipelined-based collectives outperform the kernel versions due to utilizing *copy engines*. Although not shown in the figure, this trend starts around message sizes larger than 16 MB. However, for smaller messages, the kernel versions are increasingly more efficient. This observation is consistent with previous studies [16].

IV. Conclusion and Future Work

In this paper, we presented a novel multi-path heterogeneous *Allreduce* framework for multi-GPU systems. By leveraging both GPU and CPU resources and interconnects, our approach effectively utilizes available bandwidth and computational power to enhance performance. Our experimental results demonstrate that our framework can achieve up to 1.45x speedup over NCCL. The integration of multi-path communication and heterogeneous computing demonstrates to be a promising direction for improving the efficiency of *Allreduce* operations commonly used in HPC and Deep Learning.

In future work, we will focus on extending our framework to support a larger number of GPUs and more complex topologies. This way, we can observe the applicability, scalability, and performance of our approach in multi-node systems where multiple data transfers compete for PCIe bandwidth. Additionally, we plan to explore dynamic adaptation techniques to optimize communication patterns based on applications requirements and hardware configurations. Utilizing analytical performance modeling, we aim to predict the optimal configuration for different systems and workloads. We also aim to investigate the integration of our framework within runtime libraries such as MPI and UCC. Moreover, the discussed techniques and approaches are adaptable to other collective communication operations, and lastly we will explore the applicability of our collective design in different domains such as machine learning and scientific simulations.

V. Acknowledgments

This research was supported by Digital Research Alliance of Canada. Computations were performed partly on Narval with support from Calcul Québec (calculquebec.ca) and Mist supercomputer at the SciNet HPC Consortium. SciNet is funded by Innovation, Science and Economic Development Canada; the Digital Research Alliance of Canada; the Ontario

Research Fund: Research Excellence; and the University of Toronto.

References

[1] Pedram Alizadeh, Amirhossein Sojoodi, Yiltan Hassan Temucin, and Ahmad Afsahi. Efficient Process Arrival Pattern Aware Collective Communication for Deep Learning. In *Proceedings of the European MPI Users' Group Meeting (EuroMPI)*, pages 68–78, 2022.

[2] Tsuyoshi Ichimura and Jack Wells. Heterogeneous computing in a strongly-connected CPU-GPU environment : fast multiple time-evolution equation-based modeling accelerated using data-driven approach. In *Proceedings of the workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W)*, pages 1967–1978, 2024.

[3] Weijie Liu, Kai Lu, Zhiquan Lai, Shengwei Li, Keshi Ge, Dongsheng Li, and Xicheng Lu. AutoPipe-H: A Heterogeneity-Aware Data-Paralleled Pipeline Approach on Commodity GPU Servers. *IEEE Transactions on Computers*, pages 1–14, 2024.

[4] Chris Loken, Daniel Gruner, Leslie Groer, Richard Peltier, Neil Bunn, Michael Craig, Teresa Henriques, Jillian Dempsey, Ching-Hsing Yu, Joseph Chen, L Jonathan Dursi, Jason Chong, Scott Northrup, Jaime Pinto, Neil Knecht, and Ramses Van Zon. Scinet: Lessons learned from building a power-efficient top-20 system and data centre. *Journal of Physics: Conference Series*, 256(1):012026, nov 2010.

[5] NVIDIA Collective Communications Library, 2025. https://github.com/NVIDIA/nccl [Accessed: 2025-01-15].

[6] Open MPI, 2025. https://www.open-mpi.org/ [Accessed: 2025-01-15].

[7] Marcelo Ponce, Ramses van Zon, Scott Northrup, Daniel Gruner, Joseph Chen, Fatih Ertinaz, Alexey Fedoseev, Leslie Groer, Fei Mao, Bruno C. Mundim, Mike Nolta, Jaime Pinto, Marco Saldarriaga, Vladimir Slavnic, Erik Spence, Ching-Hsing Yu, and W. Richard Peltier. Deploying a top-100 supercomputer for large parallel workloads: the niagara supercomputer. In *Practice and Experience in Advanced Research Computing 2019: Rise of the Machines (Learning)*, PEARC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[8] Junyeol Ryu, Jinpyo Kim, Heehoon Kim, Daeyoung Park, and Jaejin Lee. SPipe : Hybrid GPU and CPU Pipeline for Training LLMs under Memory Pressure. In *Proceedings of the 20th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 1–19, 2025.

[9] Thomas R.W. Scogland, Wu Chun Feng, Barry Rountree, and Bronis R. De Supinski. CoreTSAR: Adaptive worksharing for heterogeneous systems. *Lecture Notes in Computer Science (LNCS)*, pages 172–186, 2014.

[10] Amirhossein Sojoodi, Yıltan Hassan Temucin, and Ahmad Afsahi. Enhancing Intra-Node GPU-to-GPU Performance in MPI + UCX through Multi-Path Communication. In *Proceedings of the International Workshop on Extreme Heterogeneity Solutions (ExHET)*, pages 1–6, 2024.

[11] Yuya Tatsugi and Akira Nukada. Accelerating data transfer between host and device using idle GPU. In *Proceedings of the Workshop on General Purpose Processing using GPUs (GPGPU)*, pages 1–6, 2022.

[12] Yıltan Hassan Temucin, Amirhossein Sojoodi, Pedram Alizadeh, and Ahmad Afsahi. Efficient Multi-Path NVLink / PCIe-Aware UCX based Collective Communication for Deep Learning. In *Proceedings of the IEEE Symposium on High-Performance Interconnects (HOTI)*, pages 1–10, 2021.

[13] Yıltan Hassan Temucin, Amirhossein Sojoodi, Pedram Alizadeh, Benjamin W Kitor, and Ahmad Afsahi. Accelerating Deep Learning using Interconnect-Aware UCX Communication for MPI Collectives. *IEEE Micro*, pages 1–9, 2021.

[14] Unified Collective Communication (UCC), 2025. https://github.com/openucx/ucc [Accessed: 2025-01-15].

[15] Didem Unat, Ilyas Turimbetov, Mohammed Kefah, Taha Issa, Flavio Vella, Daniele D E Sensi, and Ismayil Ismayilov. The Landscape of GPU-Centric Communication. *arXiv*, pages 1–25, 2024.

[16] Guanhua Wang, Shivaram Venkataraman, Amar Phanishayee, Jorgen Thelin, Nikhil Devanur, and Ion Stoica. Blink: Fast and Generic Collectives for Distributed ML. In *Proceedings of Machine Learning and Systems (MLSys)*, pages 1–15, 2020.

[17] Liangyu Zhao, Saeed Maleki, Ziyue Yang, Hossein Pourreza, Aashaka Shah, Changho Hwang, and Arvind Krishnamurthy. ForestColl : Efficient Collective Communications on Heterogeneous Network Fabrics. *arXiv*, pages 1–26, 2024.

[18] Xuanlei Zhao, Bin Jia, Haotian Zhou, Ziming Liu, and Yang You. HeteGen: Efficient Heterogeneous Parallel Inference for Large Language Models on Resource-Constrained Devices. In *Proceedings of Machine Learning and Systems (MLSys)*, pages 1–11, 2024.