The Big Send-off: High Performance Collectives on GPU-based Supercomputers

Siddharth Singh
Department of Computer Science,
University of Maryland
College Park, USA
ssingh37@umd.edu

Mahua Singh
Dept. of Computer Science and Engg.,
Indian Institute of Technology
Guwahati, India
s.mahua@iitg.ac.in

Abhinav Bhatele
Department of Computer Science,
University of Maryland
College Park, USA
bhatele@cs.umd.edu

Abstract

We evaluate the current state of collective communication on GPU-based supercomputers for large language model (LLM) training at scale. Existing libraries such as RCCL and Cray-MPICH exhibit critical limitations on systems such as Frontier – Cray-MPICH underutilizes network and compute resources, while RCCL suffers from severe scalability issues. To address these challenges, we introduce PCCL, a communication library with highly optimized implementations of all-gather and reduce-scatter operations tailored for distributed deep learning workloads. PCCL is designed to maximally utilize all available network and compute resources and to scale efficiently to thousands of GPUs. It achieves substantial performance improvements, delivering 6–33× speedups over RCCL and 28–70× over Cray-MPICH for all-gather on 2048 GCDs of Frontier. These gains translate directly to end-to-end performance: in large-scale GPT-3-style training, PCCL provides up to 60% and 40% speedups over RCCL for 7B and 13B parameter models, respectively.

1 Introduction

In the last few years, large language models (LLMs) have resulted in significant advancements in natural language processing [6, 12, 22]. These models are extremely adept at generating and manipulating text with high fidelity and have facilitated automation in tasks such as text summarization, translation, code generation, and personal learning. At the core of these advancements are two key factors: the use of large-scale datasets for training and the development of models with billions of parameters. Both aspects are crucial for enabling LLMs to achieve their remarkable performance but come with significant computational demands. Training these models requires extensive hardware resources, often involving thousands to tens of thousands of GPUs to handle the immense computational load. For instance, LLaMA 3, a model with 405 billion parameters, was trained using 16,000 H100 GPUs [12]!

At large GPU counts, communication quickly becomes the primary bottleneck to efficient scaling. While modern GPUs leverage specialized tensor cores to accelerate matrix operations in LLMs, these speedups increase the relative cost of communication. Advanced parallel training algorithms – such as ZeRO [19] and FSDP [27] – depend heavily on collective operations like all-gather and reduce-scatter, which frequently move tens to hundreds of megabytes per call. As both the number of GPUs and message sizes grow, efficiently handling this communication becomes increasingly difficult. To support the demands of large-scale deep learning, communication libraries must therefore be highly scalable and specifically optimized for these workloads.

In this work, we focus on OLCF's Frontier, an AMD MI250X based supercomputer and evaluate the efficacy of existing communication libraries for collective performance in deep learning. Specifically, we examine all-gather and reduce-scatter collectives, which are widely used in distributed training frameworks like ZeRO and FSDP. On Frontier, users have two main choices for the communication library: Cray-MPICH, an MPICH-based implementation of MPI optimized for HPC workloads, and RCCL, AMD's ROCm Collective Communication Library designed for GPU-centric communication. We demonstrate that both libraries exhibit unique shortcomings on Frontier, leading to inefficiencies that hinder scalability and make them suboptimal for large-scale LLM training.

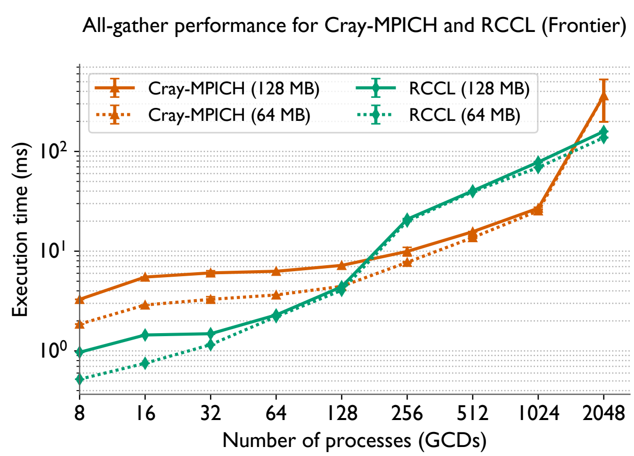

Figure 1: Performance comparison of all-gather using Cray-MPICH vs. RCCL on Frontier for two output buffer sizes of 64 and 128 MB. The ideal scaling behavior (flat horizontal line) is not achieved by either library, highlighting their limited scalability at increasing GCD counts.

Cray-MPICH fails to fully utilize the available compute and network resources on each node, sustaining only a fraction of the

,

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

system's peak bandwidth. While RCCL achieves high bandwidth at small scales, its performance deteriorates sharply as we scale to hundreds or thousands of GPUs. Figure 1 illustrates these issues by benchmarking the all-gather collective on Frontier with two output buffer sizes: 64 and 128 MB. For both message sizes, the ideal scaling behavior is a flat horizontal line. However, we observe significant performance degradation in both libraries beyond 256 processes, highlighting their limitations and making them suboptimal for large-scale training workloads.

This work introduces PCCL, the Performant Collective Communication Library, designed to accelerate collective operations - specifically all-gathers and reduce-scatters - for parallel deep learning workloads. PCCL includes highly optimized implementations tailored for message sizes in the tens to hundreds of megabytes, which are commonly encountered in large-scale training. Our design focuses on alleviating key performance bottlenecks in Cray-MPICH and RCCL by leveraging the strengths of both: harnessing the system's networking and accelerated computing resources while optimizing for latency-bound scenarios that emerge at extreme scale.

With all of these optimizations in place, our implementations of all-gather and reduce-scatter achieve significant speedups over both RCCL and Cray-MPICH. For instance, on 2048 GPUs of Frontier (256 nodes), our all-gather implementation delivers a 6–33× speedup over RCCL and a 28–70× speedup over Cray-MPICH. These improvements also seamlessly translate to end-to-end training performance. On 1024 GCDs of Frontier, replacing RCCL with PCCL for collective communication results in substantial training speedups – 60% for a 7B parameter model and 40% for a 13B parameter model. Our optimized collectives thus pave the way for scalable, high-performance training of large-scale deep learning models on next-generation GPU supercomputers.

- We analyze the limitations of existing communication libraries, Cray-MPICH and RCCL, for all-gather and reduce-scatter collectives in parallel deep learning workloads.
- We develop optimized implementations of these collectives in PCCL, with a focus on effectively utilizing system resources and ensuring scalability in latency bound scenarios.
- We conduct end-to-end benchmarking of large-scale LLM training workloads to validate the practical benefits of our optimizations, demonstrating significant speedups in training throughput.

2 Background

In this section, we provide relevant background on parallel or distributed deep learning with a focus on the role of collective communication in the functioning of state-of-the art parallel deep learning frameworks.

2.1 Parallel Deep Learning and Collective Communication

While several categories of parallelism exist in deep learning (tensor parallelism [20], pipeline parallelism [15], expert parallelism [18]), this work focus on sharded data parallelism, a widely used approach for large scale training [19, 27]. In this paradigm, model parameters and gradients are partitioned (or "sharded") across GPUs, which significantly reduces memory requirements and allows for the training of extremely large models. Two critical collective communication operations – *all-gather* and *reduce-scatter* – play a central role in sharded data parallelism. These operations aggregate distributed data across GPUs: the all-gather operation collects model parameters from all shards to form a complete copy, while the reduce-scatter operation performs a reduction and distributes gradients across participating processes. In Figure 2, we plot the all-gather and reduce-scatter message sizes for three frameworks that support sharded data parallelism – FSDP [27], Deepspeed ZeRO-3 [19], and AxoNN [21]. Notice how the message sizes across these three frameworks are in the tens to hundreds of megabytes, even becoming more than a gigabyte for larger models.

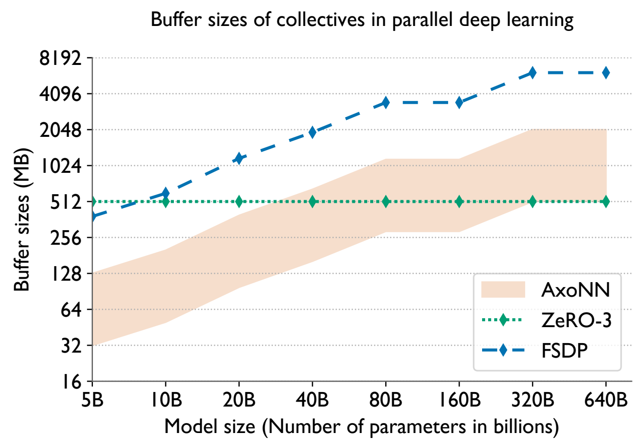

Figure 2: Distribution of all-gather and reduce-scatter message sizes for several deep learning frameworks for a range of transformer [25] model sizes. The y-axis represents input buffer sizes for all-gathers but output buffer sizes for reduce-scatters.

2.2 Algorithms for All-Gathers and Reduce-Scatters

Efficient implementations of all-gather and reduce-scatter operations are critical for sharded data parallelism. In this work, we build on several well-established algorithms and introduce enhancements to improve performance and scalability.

Ring: The ring algorithm is a popular method for implementing collective communications due to its simplicity and efficiency in certain network topologies. In a ring-based all-gather or reduce-scatter, each process communicates with its immediate neighbors in a circular fashion. While effective at moderate scales and large message sizes, the ring algorithm can suffer from inefficiencies at larger scales due to its latency term being linearly proportional to the number of processes. For example, the communication time of a ring all-gather can be modeled as

$$T_{\text{ring}} = \alpha \times (p-1) + \beta \times \frac{p-1}{p} m$$

where p is the number of processes, α represents the startup latency per message, m is the size of the input buffer on each GPU, and β is the inverse of the peer-to-peer bandwidth.

Recursive Halving/Doubling: A popular way of minimizing latency costs involves utilizing the recursive halving or doubling algorithms for all-gathers and reduce-scatters respectively. These algorithms divide the communication task into logarithmically many steps, and hence their performance scales better than the ring algorithm. For example, the communication time for a recursive-halving all-gather can be modeled as

$$T_{\text{rec}} = \alpha \times \log_2(p) + \beta \times \frac{p-1}{p} m$$

where the terms are the same as before. For small message sizes or very large process counts, the logarithmic growth in the latency term often leads to lower overall communication costs. More details about these and other algorithms can be found in Thakur et al. [23].

3 Current State of Cray MPICH and RCCL

As established in the previous section, all-gather and reduce-scatter operations are important collectives in parallel deep learning [19, 21, 27]. Thus, to scale model training to the thousands of GPUs required for large models, we need highly efficient and scalable implementations of these collectives – particularly for the large message sizes characteristic of deep learning workloads (see Figure 2). This section investigates the state of the current state of the practice of popular communication libraries - Cray-MPICH and RCCL, for these collectives. We find unique issues that plague the performance of each library, and we highlight these below via experiments on the Frontier supercomputer.

3.1 Benchmarking Methodology

First, let us look at the methodology we used to benchmark the performance of the two libraries on Frontier.

Process Placement and NUMA Configuration: Each node on Frontier is equipped with four GPUs, which are each partitioned into two Graphic Compute Dies (GCDs). We therefore launch one MPI process per GCD, binding each process to seven CPU cores and leaving two cores per NUMA region free for operating-system tasks and to minimize noise. To ensure NUMA-aware placement of network interfaces, we configure the OFI provider with `MPICH_OFI_NIC_POLICY=USER` and `MPICH_OFI_NIC_MAPPING="0:0-1;1:2-3;2:4-5;3:6-7"`, which pins each MPI rank to the appropriate NIC ports in its NUMA domain.

Message Sizes and Measurement Protocol: In line with Figure 2, our evaluation focuses on message sizes from 16MB up to 1GB. Note that for all-gathers and reduce-scatters, these values refer to the output and input message size per GPU respectively. For each combination of library, collective, message size, and GPU count, we perform ten independent runs. We measure the total time spent in the collective on each run using AMD's `hipeventtimers` instrumentation and computed the mean and standard deviation over the ten runs to ensure statistical robustness.

Communication Tuning: We disable all forms of eager messaging in the OFI/CXI provider by setting `FI_CXI_RDZV_THRESHOLD=0`, `FI_CXI_RDZV_GET_MIN=0`, and `FI_CXI_RDZV_EAGER_SIZE=0`. For the message sizes we target, disabling eager messaging significantly improves collective performance for both Cray MPICH and RCCL. We enable GPU Direct RDMA for GPUs and NICs sharing the same NUMA node via `NCCL_NET_GDR_LEVEL=PHB` and disable HSA's SDMA engine by setting `HSA_ENABLE_SDMA=0`, ensuring that data transfers bypass the host.

Software Stack: Our software stack comprises of ROCm 6.2.4, RCCL 2.20.5, Cray MPICH 8.1.31 distribution, libfabric 1.15.2 and the aws-ofi-rccl plugin version v1.4.

3.2 Poor MPI performance at lower GPU counts

Figure 3 (left) presents a comparative analysis of all-gather performance between Cray MPICH and RCCL on Frontier, specifically for large message sizes of 256 MB and 512 MB. Despite both libraries implementing a ring-based collective algorithm over the OpenFabrics Interfaces (OFI) layer, RCCL achieves approximately a 4× performance advantage in this bandwidth-bound scenario. To explain this disparity, we examine hardware performance counters provided by the Cassini Slingshot-11 Network Interface Controllers (NICs) [7] on each Frontier node.

Our investigation focuses on the counters `parbs_tarb_pi_posted_pkts` and `parbs_tarb_pi_non_posted_pkts`, which, based on our understanding, represent the count of packets read from and written to each NIC within a node during job execution. The middle and right plots of Figure 3 demonstrate a significant divergence in NIC utilization between the two libraries. Cray MPICH constrains all read operations to NIC 3 and all write operations to NIC 0, effectively creating a single-NIC bottleneck. Conversely, RCCL distributes network traffic more uniformly across all available NICs. This equitable load distribution leads to enhanced bandwidth utilization and a substantial reduction in all-gather execution time. This observed imbalance in NIC utilization directly accounts for the pronounced performance gap between Cray MPICH and RCCL in this bandwidth-bound context.

> **Explanation 1**
>
> Cray MPICH routes all network traffic through a single NIC, resulting in severe underutilization of the available network bandwidth. In contrast, RCCL effectively balances node traffic across all four NICs, achieving a four-fold performance improvement over Cray MPICH.

3.3 Poor Performance of MPI_Reduce_scatter

Next, we examine reduce-scatter performance. As shown in Figure 4, Cray MPICH (orange) performs significantly worse than RCCL (green). Notably, this performance gap is far greater than the 4× difference observed earlier for all-gather in Figure 3. While the NIC underutilization issue outlined in the previous subsection still persists for Cray-MPICH Reduce-scatters, it alone cannot explain this performance disparity. We hypothesize that this disparity stems from the way reduction computations are scheduled in the two libraries. Cray MPICH performs the reduction operations required

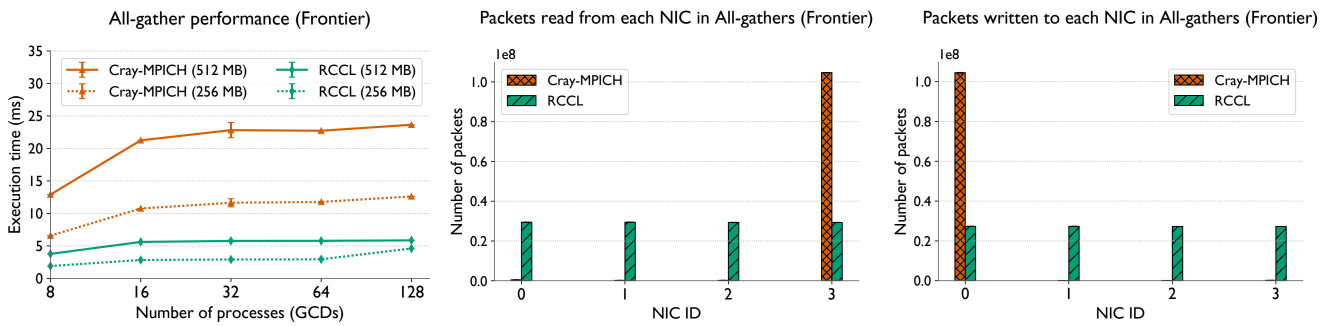

Figure 3: The left plot compares all-gather performance of Cray MPICH and RCCL on Frontier for a bandwidth-bound scenario with large message sizes (256 and 512 MB) and small GPU counts. The middle and right plot show the number of packets read from (left) and written to (right) each of the four NICs on a Frontier compute node during all-gather operations.

for reduce-scatter on the CPU, introducing significant computational overheads for large messages. In contrast, RCCL efficiently performs these operations by offloading them to the GPUs, leveraging their parallel processing capabilities.

To test this hypothesis, we manually implemented the reduce-scatter operation using Cray MPICH point-to-point sends and receives, while scheduling the reduction operations on the GPU via a HIP vector-addition kernel. As shown in Figure 4, our implementation (blue line) achieves performance that is several times faster than Cray MPICH's native reduce-scatter, further supporting our hypothesis.

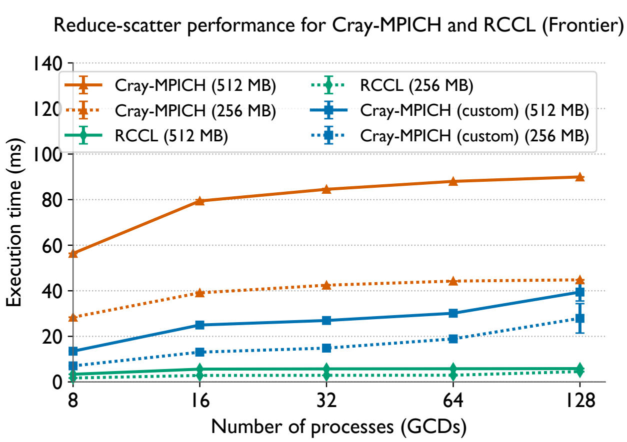

Figure 4: Performance comparison of reduce-scatter using Cray MPICH, RCCL, and a custom implementation of reduce-scatter that uses Cray MPICH P2P and GPU compute kernels.

Explanation 2

Cray MPICH's CPU-based reduction operations in reduce-scatter introduce significant overhead, which in combination with the NIC underutilization issue, results in a 10-15x performance gap compared to RCCL's GPU-accelerated reductions.

3.4 Poor Scaling of RCCL and MPI at Large GPU Counts

Figure 1 shows all-gather performance for Cray MPICH and RCCL when sending relatively small messages across increasing numbers of GCDs. We observe that both libraries exhibit poor scaling behavior at large GPU counts. On investigating deeper, we found that both Cray MPICH and RCCL only support the ring algorithm for all-gathers and reduce-scatters (see Section 2.2). While effective for bandwidth-bound workloads, the ring algorithm performs poorly in latency-bound scenarios because each process must send and receive $(p-1)$messages sequentially, causing the total communication time to grow linearly with the number of processes.

Curiously, neither library implements more optimal algorithms like recursive doubling or halving (see Section 2.2), which are known to reduce the number of communication steps to $\log_2 p$ and are generally preferred for small message sizes or high process counts. This lack of algorithmic diversity directly contributes to the sub-optimal scaling we observe at large GPU counts.

Explanation 3

Both Cray MPICH and RCCL rely solely on the ring algorithm for all-gather and reduce-scatter, leading to poor scaling inlatency-bound scenarios. More efficient algorithms like recursive doubling and halving are not supported.

4 Optimizing All-gathers and Reduce-scatters

In Section 3, we identified several challenges affecting RCCL and Cray-MPICH in the context of all-gather and reduce-scatter collectives for deep learning workloads. These challenges create significant barriers to efficiently scaling large model training across thousands of GPUs. The central theme of this work is to developed optimized implementations of all-gather and reduce-scatter collectives that address these challenges. In this section, we present our proposed solutions, which are implemented in a new library called PCCL (Performant Collective Communication Library). We begin by discussing the design principles and strategies that drive our proposed solutions.

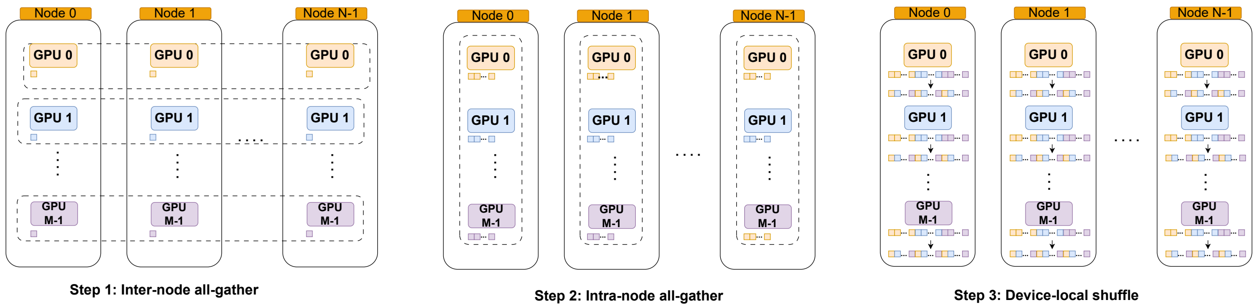

Figure 5: Diagram showing our hierarchical (two-level) implementation to dissolve an all-gather operation on a GPU-based cluster with N nodes and M GPUs per node. In Step 1, we performs inter-node all-gathers, in step 2, we perform intra-node all-gathers and in step 3, each GPU performs a local shuffle of the received data.

4.1 Hierarchical Collective Algorithms for Load Balancing NIC Traffic

Our optimized implementations of all-gather and reduce-scatter are based on a two-level hierarchical design. While prior work has demonstrated that hierarchical algorithms can reduce latency and improve scalability in collective operations [3, 24], our primary motivation for adopting this design is to address the NIC underutilization problem identified in Section 3.2. Now, we provide a brief overview of the inner workings of our hierarchical design.

We illustrate our design in Figure 5 for an all-gather operation on a hypothetical system with N nodes and M GPUs per node. The global collective operation is divided into two distinct phases using sub-communicators: inter-node sub-communicators and intra-node sub-communicators. Inter-node sub-communicators are formed by grouping corresponding GPUs across nodes in a group. For example, in Figure 5, all GPUs with the same index across nodes are grouped together to form a total of M inter-node sub-communicators. Similarly, intra-node sub-communicators are formed by grouping together all GPUs within a node.

The hierarchical communication unfolds in three steps. First, in the inter-node all-gather phase, we schedule an all-gather operation in all of the inter-node sub-communicators. This is illustrated in Step-1 of Figure 5. Once this phase is completed, each GPU in a node has received data from its corresponding GPU in the other nodes. Now, within every node we have the entire result of the all-gather operation, but the data is split across GPUs. Therefore, the next step is to perform an intra-node all-gather operation, which is illustrated in Step-2 of Figure 5. Once this phase is complete, each GPU now has the complete output in its memory, albeit in an incorrect order. So, the final step involves a device-local shuffle operation, where each GPU rearranges its data to put in a correct order. This is illustrated in Step-3 of Figure 5. The device-local shuffle is performed using a transpose kernel in practice. We implement reduce-scatter in a similar manner – but starting with the intra-node phase first, followed by the inter-node phase.

Having explained the workings of our hierarchical design, let us now see how it addresses the NIC underutilization problem. An important aspect of our design in that it schedules all of the all-gather operations in Step-1 of Figure 5 concurrently on all of the inter-node sub-communicators. We leverage this fact to utilize all NICs on a node concurrently. A Frontier node has four NICs, each connected to two GCDs. In our implementation, we ensure that each GCD exclusively sends and receives traffic to and from its corresponding NIC (e.g. - GCDs 0 and 1 to NIC 0, GCDs 2 and 3 to NIC 1, and so on). This is how we ensure that the inter-node traffic is evenly distributed across all NICs in PCCL.

4.2 Choice of Communication Libraries for Each Level of the Hierarchy

We now describe the choice of communication libraries we make for each level of the hierarchy, starting with the intra-node level.

GPU-vendor libraries like RCCL are highly optimized for intra-node topologies, efficiently utilizing shared memory, PCIe, and Infinity fabric connections. These optimizations significantly outperform most MPI implementations in managing GPU-to-GPU communication within a node [5]. Hence, we simply rely on RCCL for all of our communication in the intra-node phase.

Prior work has reported that RCCL is not robust at scale and can crash during training runs [9]. This issue has also been noted by HPE[1] and in the OLCF User Guide[2]. Thus for reliability reasons, we opt to use Cray-MPICH for all inter-node communication.

4.3 Choice of Algorithms for Inter-Node Communication

Our choice of communication algorithms for each level of the hierarchy is driven by performance considerations and the limitations of available libraries. Since RCCL only supports the ring algorithm for intra-node collectives, we adopt this as our intra-node communication strategy. Fortunately, ring is well-suited for this context, as the small number of GCDs within a node (eight) ensures that ring can effectively saturate the available bandwidth.

The inter-node phase, however, presents greater challenges. With potentially thousands of GPUs participating in the collective, latency concerns become critical. Cray-MPICH, which we rely on for inter-node communication due to RCCL's instability at scale, offers only the ring algorithm by default. Unfortunately, as described in

[1]https://www.olcf.ornl.gov/wp-content/uploads/OLCF_AI_Training_0417_2024.pdf
[2]https://docs.olcf.ornl.gov/software/analytics/pytorch_frontier.html#environment-variables

Section 2.2 ring algorithms's linear scaling in latency with respect to the number of processes makes it suboptimal at large-scale.

To address this, we implement alternative algorithms with improved scaling properties. Specifically, we utilize recursive doubling for all-gather operations and recursive halving for reduce-scatter operations [23]. These algorithms offer logarithmic latency terms (see Section 2.2), enabling significantly better performance as the number of GPUs increases. Our implementations are based on Cray-MPICH's point-to-point send and receive operations. Moreover, for reduce-scatter operations, we also ensure that our vector addition computation is efficient by scheduling it on the GPU cores.

In Figure 6 (left), we demonstrate the speedup of using recursive halving over ring in the inter-node phase of our reduce-scatters. Note that both of these implementations use RCCL's ring algorithm for the intra-node phase. We observe that ring is the preferred choice of algorithm for inter-node communication in bandwidth bound scenarios (smaller process counts and/or larger message sizes). However, as expected recursive halving becomes the more optimal algorithm for latency bound scenarios (larger process counts and/or small message sizes).

We develop these implementations in C++ as part of PCCL and expose Pybind11 bindings to enable seamless integration with Python-based deep learning frameworks such as ZeRO-3 [19]. Implementing these algorithms in C++ proves to be critical for achieving high performance. As shown in Figure 6 (right), a baseline implementation using Python and mpi4py can be nearly 4× slower than our optimized C++ version—compare the performance at 1024 GCDs. This highlights the importance of minimizing CPU-side overhead and reducing language-level inefficiencies for large-scale collective communication.

5 Experimental Setup

In this section, we provide an overview of our empirical evaluation of PCCL, the communication library proposed in this work, on state-of-the-art multi-GPU supercomputers. We conduct our experiments on the Frontier supercomputer at Oak Ridge National Laboratory and the Perlmutter supercomputer at the National Energy Research Scientific Computing Center. First, we compare the performance of all-gather and reduce-scatter operations, which are the primary focus of this paper. As mentioned in Section 2, these collective communication primitives are critical for the scalability of distributed deep learning workloads. Second, we evaluate the end-to-end performance of a parallel deep learning framework - DeepSpeed ZeRO-3 [19], to demonstrate the practical impact of PCCL in real-world training scenarios of multi-billion parameter models at scale.

5.1 Comparing Performance of Collectives

Our experiments cover a range of message sizes from 16MB to 1GB. For reduce-scatter, this range represents the size of the input buffer on each GPU, while for all-gather, it corresponds to the size of the output buffer on each GPU. For each message size, we measure performance across 32 to 2048 GCDs (4 to 256 nodes) on Frontier, and 32 to 2048 GPUs (8 to 512 nodes) on Perlmutter. We use HIP and CUDA event timers to measure the runtime of collective operations. On Frontier, our setup for process placement, communication tuning, and software stack is consistent with the configuration described in Section 3.1. On Perlmutter, we adopt a similar strategy by launching one process per GPU, mapping each GPU to its nearest NIC, and disabling eager communication in the FI backend. The software stack used on Perlmutter comprises of `NCCL 2.24.3`, `CUDA 12.4`, and `Cray-MPICH 8.1.30` in our experiments.

5.2 Comparing End-to-End Training Performance

To evaluate the practical benefits of PCCL, we measure the end-to-end training performance of large language models using DeepSpeed ZeRO-3 [19], a widely adopted parallel deep learning framework. We perform strong scaling experiments on 7B and 13B parameter GPT-style transformer models [1] using model hyperparameters from Zhang et al. [26]. We list these hyperparameters in Table 1. We use a global batch size of 4 million tokens and a sequence length of 2048 and use the OpenWebText [10] corpus to create our training data.

Table 1: Architectural details of the GPT-style transformer models [1] used in the experiments. We borrow these hyperparameters from Zhang et al. [26].

Model	# Parameters	# Layers	Hidden-Size	# Heads
GPT-7B	7B	32	4096	32
GPT-13B	13B	40	5120	40

On Frontier, we scale from 128 to 1024 GCDs, and on Perlmutter, we scale from 256 to 2048 GPUs. We first run ZeRO-3 with RCCL on Frontier and NCCL on Perlmutter, which represent the default communication libraries used by most parallel training frameworks. We then swap in PCCL to handle all-gather and reduce-scatter operations and rerun the experiments to observe its impact on training performance. For each configuration, we run 10 training batches across three trials and compute the average throughput over the last 8 batches in each run to minimize warm-up effects.

6 Results

We now present and analyze the results of the empirical experiments described in Section 5.

6.1 Performance Improvements Using PCCL

We begin by examining the performance of PCCL for all-gather and reduce-scatter operations and comparing it against other state-of-the-art communication libraries.

6.1.1 Comparison with Cray-MPICH and RCCL on Frontier. Let us start with examining PCCL's performance on the Frontier supercomputer. Figure 7 shows the performance of all-gather operations on Frontier using PCCL and other communication libraries. We evaluate two sets of output buffer sizes: 64 and 128 MB (left plot), and 256 and 512 MB (right plot). For each configuration, we scale the number of GCDs from 32 to 2048. Since the output buffer size per GPU remains fixed, the ideal performance curve for each buffer size is a flat horizontal line, indicating perfect scaling.

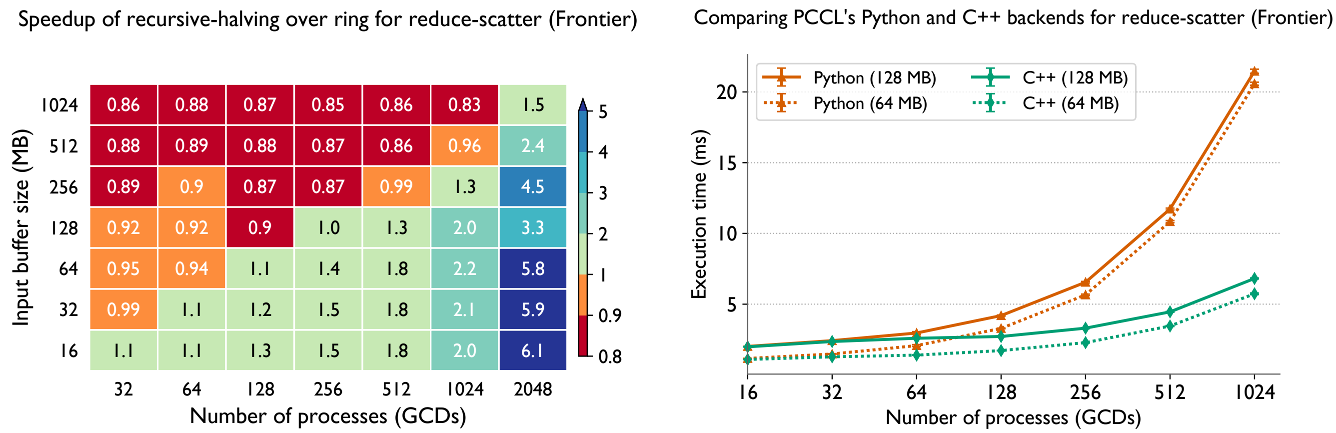

Figure 6: (Left) Heatmap showing speedups from using recursive halving over the ring algorithm in the inter-node phase of the reduce-scatter implementation in PCCL, and (Right) Performance comparison of the C++ (with Pybind11) and Python based implementations of reduce-scatter in PCCL.

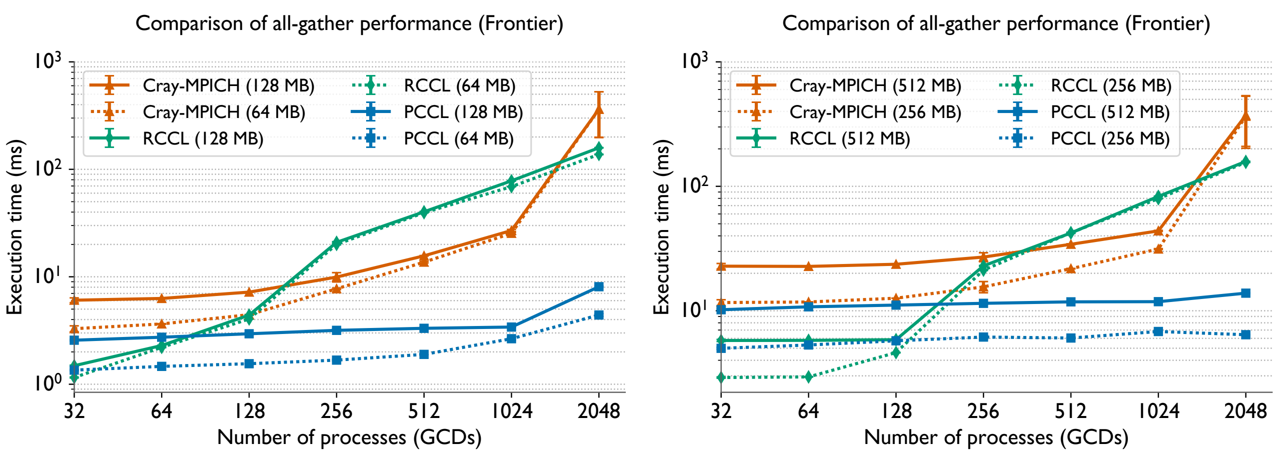

Figure 7: Performance comparison of all-gather using Cray MPICH, RCCL, and PCCL, for different per-process output buffer sizes (left plot: 64 and 128 MB, right plot: 256 and 512 MB) and varying process counts on Frontier.

However, we observe that RCCL and Cray-MPICH fall short of this ideal. For smaller message sizes in the left plot, RCCL (green lines) scales poorly, with execution time increasing almost linearly with the number of processes. For the larger message sizes in the right plot, RCCL performs well up to 128 processes, but experiences significant degradation beyond that—mirroring the trends seen with smaller messages. Cray-MPICH (orange lines) shows a similar pattern, with performance dropping sharply as we scale to higher process counts. We attribute the poor scaling of RCCL and Cray-MPICH to their reliance on the ring algorithm, whose latency term grows linearly with the number of processes.

In contrast, PCCL (blue lines) maintains nearly flat scaling trends across all message sizes in both plots, demonstrating significantly better scalability and efficiency. We attribute PCCL's better performance to its hybrid strategy, described in Section 4, which exploits the heterogeneous network topology. By using the ring algorithm within nodes (limited to eight processes) and recursive doubling across nodes, PCCL bounds the latency overhead that otherwise grows linearly in traditional ring-based implementations. This design enables better scalability across large GPU counts. The performance improvements of PCCL over RCCL and Cray-MPICH become increasingly pronounced as we increase the number of processes (GCDs). At 2048 processes, PCCL achieves speedups ranging from 7 – 24× over RCCL, and an even larger 27 to 82× over Cray-MPICH, depending on the message size. These results highlight PCCL's ability to deliver high-performance communication at scale.

Next, let us examine reduce-scatter performance on Frontier, as shown in Figure 8. Again, we observe similar trends as in the case of all-gather. Both RCCL and Cray-MPICH fall short of the ideal performance curve, and PCCL acheives significant speedups over both libraries with increasing scale.

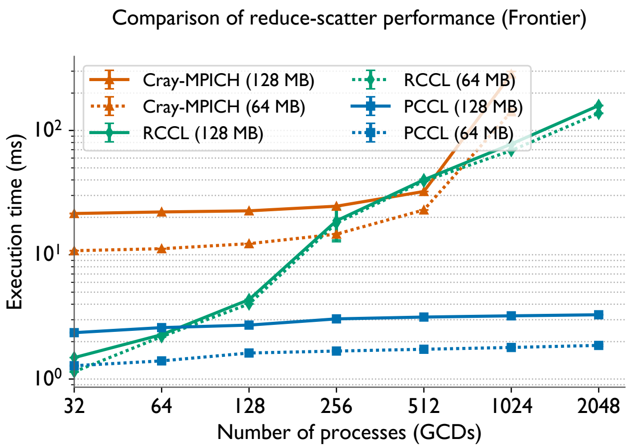

Figure 8: Performance comparison of reduce-scatter using Cray MPICH, RCCL, and PCCL, for different per-process input buffer sizes (left plot: 64 and 128 MB, right plot: 256 and 512 MB) and varying process counts on Frontier.

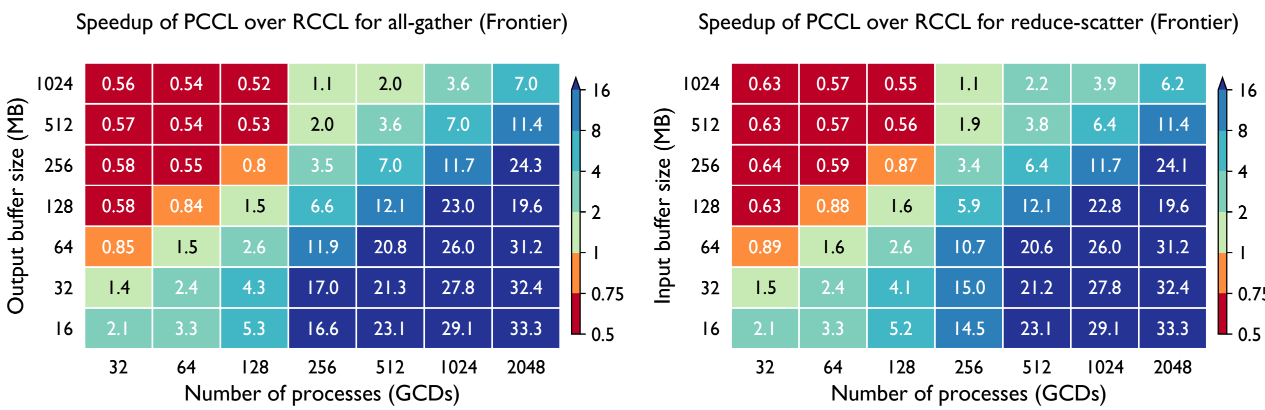

Figure 9: Heatmaps showing speedups from using PCCL over RCCL for all-gather (left) and reduce-scatter (right) on Frontier. The speedup is shown as a function of per-process output/input buffer size (in MB) and process count.

Since RCCL is the default library used by most distributed deep learning applications on AMD platforms, let us take a closer look at how PCCL compares against it. Figure 9 shows the speedups of PCCL over RCCL for all-gather (left) and reduce-scatter (right) operations on Frontier, respectively, across a range of output buffer sizes and process counts. In the top-left regions of both heatmaps–large messages and small GPU counts, which represent bandwidth-bound scenarios–PCCL underperforms RCCL. For instance, with a 1024 MB buffer at 32 GCDs, speedups are around 0.52× for all-gather and 0.55× for reduce-scatter. This is expected, as RCCL's flat ring algorithm can theoretically achieve higher bandwidth than PCCL's hierarchical two-phase strategy [3].

However, in the bottom-right corners–small messages and large GPU counts, where latency dominates–PCCL delivers substantial gains. For both collective operations at 2048 GCDs, PCCL achieves speedups of more than 30× over RCCL for 16MB, 32MB, and 64MB message sizes, respectively! In contrast, for larger message sizes like 512MB and 1024MB, the speedups are smaller but still significantly high —- 11.4 and 7× for all-gather, and 11.4 and 6.2× for reduce-scatter. These results underscore PCCL's strength in latency-bound scenarios and highlight its ability to scale efficiently to thousands of GPUs.

6.1.2 Comparison with Cray-MPICH and NCCL on Perlmutter. We now evaluate PCCL's effectiveness on Perlmutter, which features NVIDIA A100 GPUs. Figure 10 presents results for all-gather (left) and reduce-scatter (right) operations with message sizes of 64MB and 128MB–representing the output buffer sizes for all-gather and input buffer sizes for reduce-scatter, respectively. We observe similar trends as on Frontier. Both Cray-MPICH (orange lines) and NCCL (black lines) fall short of ideal scaling, which would appear as a flat horizontal line. Like RCCL on Frontier, NCCL's performance begins to degrade noticeably beyond 128 processes. In contrast, PCCL scales nearly perfectly across both collectives, maintaining desirable flat performance curves and achieving speedups in the range of 1.3 – 4.6× over NCCL and 8.8–15× over Cray-MPICH on 1024 and 2048 GPUs!

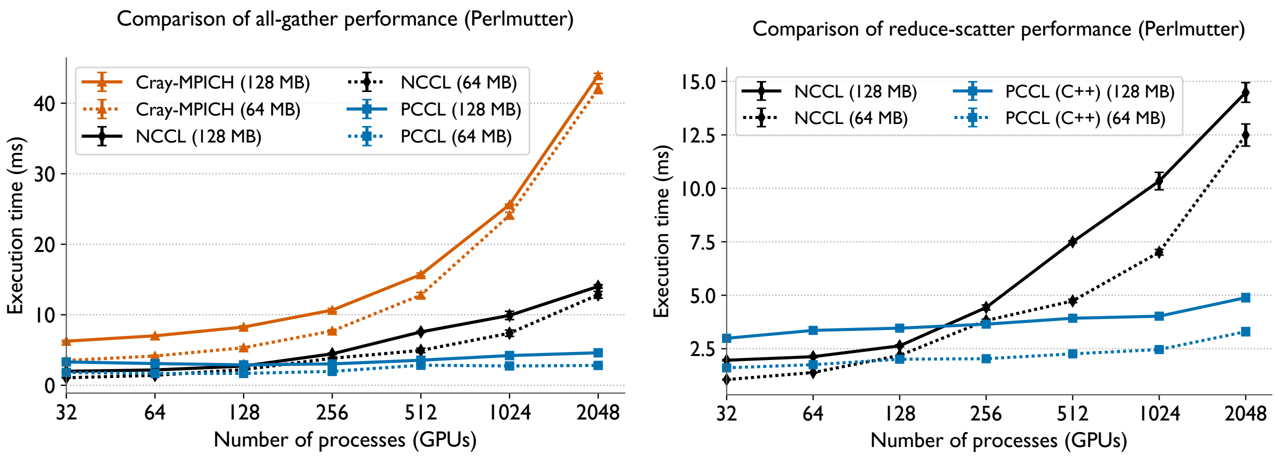

Figure 10: Performance comparison of all-gather (left plot) and reduce-scatter (right plot) using Cray MPICH, NCCL, and PCCL, for two per-process buffer sizes (64 and 128 MB) and varying process counts on Perlmutter.

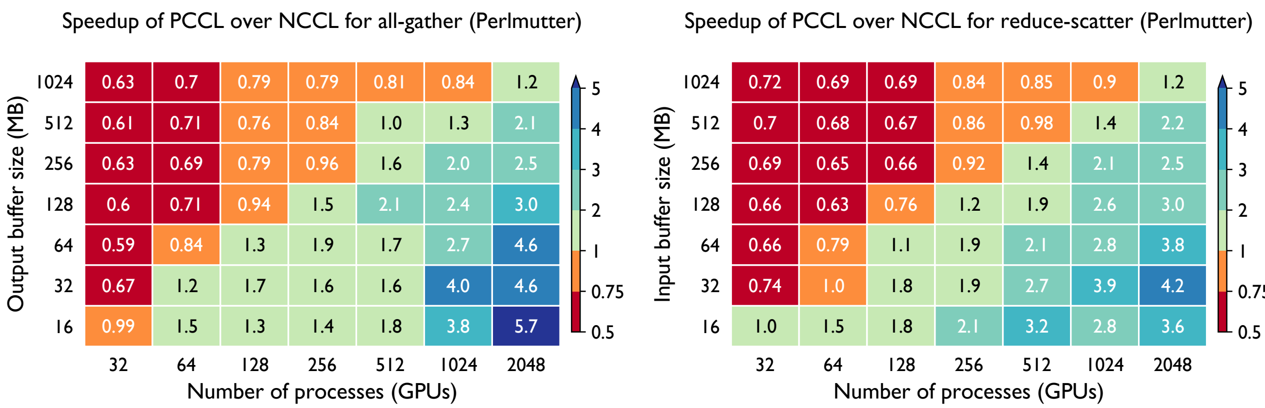

Figure 11: Heatmaps showing speedups from using PCCL over NCCL for all-gather (left) and reduce-scatter (right) on Perlmutter. The speedup is shown as a function of per-process output/input buffer size (in MB) and process count.

Figure 11 examines how PCCL compares to NCCL across various message sizes and GPU counts. Similar to RCCL, NCCL outperforms our library in the top-left regions of the heatmaps, representing bandwidth-bound scenarios. For instance, with 32 processes and a 1024 MB message size, NCCL is nearly 1.5× faster than PCCL. However, as we transition to latency-bound regions in the bottom-right corners of the heatmap, PCCL's advantages become evident. Around 1024–2048 processes and 16–32MB message sizes, PCCL achieves significant speedups over NCCL, ranging from 3–5×. While speedups for larger message sizes are smaller, they remain notable. For example, at 2048 processes and 128–512 MB message sizes, PCCL is approximately 2–3× faster than NCCL. These results highlight PCCL's effectiveness in accelerating collective communication for parallel deep learning – across both extreme scales and diverse GPU architectures.

6.2 Impact on DL Applications' Performance

Finally, we examine how these communication gains translate into improvements in end-to-end training performance at scale. The left panel of Figure 12 presents the batch times for strong scaling GPT-3-style transformer training on Frontier using the DeepSpeed ZeRO-3 framework [19]. Green lines represent ZeRO-3 runs with RCCL, the default communication library and blue lines represent runs with all-gather and reduce-scatter collectives in ZeRO-3 issued with PCCL. At smaller scales (128 and 256 GCDs), both libraries perform comparably. However, as we scale further, PCCL begins to outperform RCCL. At 512 GCDs, PCCL reduces batch time by nearly 30% for the 7B model and by 16% for the 13B model. When scaling to 1024 GCDs, RCCL fails to maintain strong scaling and even exhibits increased batch times compared to 512 GCDs. In contrast, PCCL continues to scale efficiently, delivering a 60% speedup for the 7B model and a 39% speedup for the 13B model. Finally, at 2048 GCDs, although both libraries show diminishing returns in

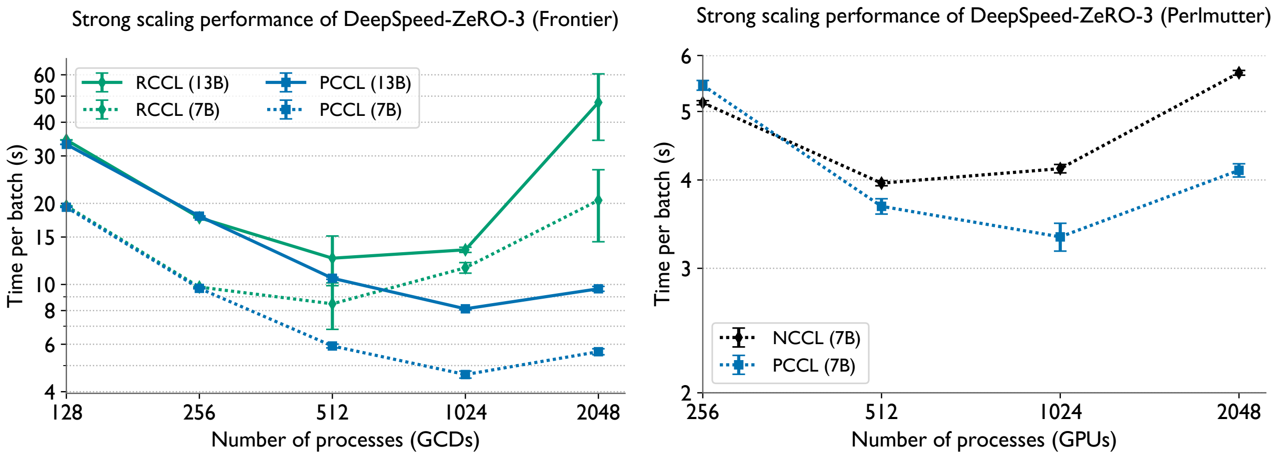

Figure 12: Strong scaling performance of Deepspeed ZeRO-3 using RCCL, NCCL, and PCCL, on Frontier (left) and Perlmutter (right) for two model sizes: GPT-3 7B and GPT-3 13B.

strong scaling efficiency, PCCL still achieves substantial speedups (70–80%) relative to RCCL.

We observe similar trends on Perlmutter, as shown in the right panel of Figure 12. At 256 GPUs, NCCL outperforms PCCL by approximately 6%. However, as we scale to larger GPU counts, PCCL begins to outperform NCCL–achieving a 7% speedup at 512 GPUs and a significantly higher 20% speedup at 1024 GPUs. All of these results highlight PCCL's ability to deliver performance improvements for collective communication across multiple GPU architectures, and more importantly, translate those gains into meaningful end-to-end speedups for large-scale deep learning applications.

7 Related Work

Optimizing collective communication has been a long-standing challenge in high-performance computing (HPC) and parallel computing and has been the focus on extensive research. Thakur et al.'s seminal work focuses on optimizing a plethora of collective operations including all-gathers and reduce-scatters in MPICH [23]. The authors explore the design space of several algorithms for each collective, and provide guidelines for selecting the most approporiate algorithm for different scenarios. In contrast, our work focuses on large-scale deep learning workloads and with messages sizes in the tens to hundreds of megabytes. Patarsuk et al. propose the bandwidth-optimized ring algorithm for all-reduce operations [17]. Graham et al. develop optimize MPI collective to effectively exploit shared memory on multi-core systems. Chan et al. citechan2006simultaneouscollectives develop highly optimized collectives for the IBM Blue Gene/L, exploiting unique properties of the system [11]. Kandalla et al. develop a scalable multi-leader hierarchical algorithm for all-gather [16]. Note that the focus of these works is on optimizing the performance of collective communication in traditional HPC workloads with small message sizes, and not on the large message sizes inherent to deep learning.

De Sensi et al. study the performance of NCCL, RCCL and Cray-MPICH across various state of the art supercomputers across a variety of latency and bandwidth bound scenarios [5]. Cho et al. propose a mutli-level hierarchical ring algorithm for all-reduce and study the tradeoff of bandwidth and latency between the flat and hierarchical ring algorithms [3]. In this work, we build on this and exploit more latency optimal algorithms like recursive doubling/halving in the inter-node levels of the hierarchy and also demonstrate how this design can be utlized to load-balance network traffic across NICs. Note that similar hierarchical designs have been explored in other works as well [13, 24]. Cai et al. develop a systematic theoretical approach to synthesize novel communication algorithms for optimizing collective communication on a particular topology [2]. Cho et al. develop a strategy to maximize the overlap of a tree-based all-reduce with the computation in neural network training [4]. There is also a body of work focused on exploiting data compression to minimize communication overheads in distributed deep learning. For example, Feng et al. optimize all-to-all communication in recommendation model training via a novel error-bounded compression algorithm [8]. Huang et al. develop hZCCL, a communication library that enables collective operations on compressed data [14]. Zhou et al. develop a GPU-based compression scheme for all-gathers and reduce-scatters [28] and optimize FSDP [27] training at scale.

8 Conclusion

In this work, we investigated the current state of collective communication libraries for parallel deep learning. Specifically, we focused on the all-gather and reduce-scatter collectives, which find wide usage in several distributed training frameworks, and comprise the bulk of communication time in large scale training runs. We evaluated the performance of Cray-MPICH, RCCL, and NCCL, three state-of-the-art-communication libraries on HPE Cray supercomputers, and highlighted shortcomings in their performance which make then unsuitable for scaling. We then developed PCCL, a new communication library with highly optimized implementations of all-gather and reduce-scatter operations. PCCL leverages several optimizations designed to alleviate the performance bottlenecks of Cray-MPICH, RCCL and NCCL, highlighted in this work. We demonstrated significant performance improvements over all three libraries, both in collective communication benchmarks as well as end-to-end training benchmarks. PCCL achieves substantial performance improvements, delivering 6–33× speedups over RCCL and 28–70× over Cray-MPICH for all-gather on 2048 GCDs of Frontier. These gains translate directly to end-to-end performance: in large-scale GPT-3-style training, PCCL provides up to 60% and 40% speedups over RCCL for 7B and 13B parameter models on Frontier, respectively.

References

[1] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. *CoRR* abs/2005.14165 (2020). arXiv:2005.14165 https://arxiv.org/abs/2005.14165

[2] Zixian Cai, Zhengyang Liu, Saeed Maleki, Madanlal Musuvathi, Todd Mytkowicz, Jacob Nelson, and Olli Saarikivi. 2021. Synthesizing optimal collective algorithms. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming* (Virtual Event, Republic of Korea) *(PPoPP '21)*. Association for Computing Machinery, New York, NY, USA, 62–75. doi:10.1145/3437801.3441620

[3] Minsik Cho, Ulrich Finkler, David Kung, and Hillery Hunter. 2019. BlueConnect: Decomposing All-Reduce for Deep Learning on Heterogeneous Network Hierarchy. In *Proceedings of Machine Learning and Systems*, A. Talwalkar, V. Smith, and M. Zaharia (Eds.), Vol. 1. 241–251. https://proceedings.mlsys.org/paper_files/paper/2019/file/0c8abcf158ed12d0dd94480681186fda-Paper.pdf

[4] Sanghun Cho, Hyojun Son, and John Kim. 2023. Logical/Physical Topology-Aware Collective Communication in Deep Learning Training. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 56–68. doi:10.1109/HPCA56546.2023.10071117

[5] Daniele De Sensi, Lorenzo Pichetti, Flavio Vella, Tiziano De Matteis, Zebin Ren, Luigi Fusco, Matteo Turisini, Daniele Cesarini, Kurt Lust, Animesh Trivedi, Duncan Roweth, Filippo Spiga, Salvatore Di Girolamo, and Torsten Hoefler. 2024. Exploring GPU-to-GPU Communication: Insights into Supercomputer Interconnects. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Atlanta, GA, USA) *(SC '24)*. IEEE Press, Article 33, 15 pages. doi:10.1109/SC41406.2024.00039

[6] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[7] Hewlett Packard Enterprise. 2024. *HPE Cassini Performance Counters*. https://cpe.ext.hpe.com/docs/latest/getting_started/HPE-Cassini-Performance-Counters.html Accessed: 2025-02-24.

[8] Hao Feng, Boyuan Zhang, Fanjiang Ye, Min Si, Ching-Hsiang Chu, Jiannan Tian, Chunxing Yin, Summer Deng, Yuchen Hao, Pavan Balaji, Tong Geng, and Dingwen Tao. 2024. Accelerating Communication in Deep Learning Recommendation Model Training with Dual-Level Adaptive Lossy Compression. arXiv:2407.04272 [cs.LG] https://arxiv.org/abs/2407.04272

[9] Jonas Geiping, Sean McLeish, Neel Jain, John Kirchenbauer, Siddharth Singh, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, and Tom Goldstein. 2025. Scaling up Test-Time Compute with Latent Reasoning: A Recurrent Depth Approach. arXiv:2502.05171 [cs.LG] https://arxiv.org/abs/2502.05171

[10] Aaron Gokaslan and Vanya Cohen. 2019. OpenWebText Corpus. http://Skylion007.github.io/OpenWebTextCorpus.

[11] Richard L. Graham and Galen Shipman. 2008. MPI Support for Multi-core Architectures: Optimized Shared Memory Collectives. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Alexey Lastovetsky, Tahar Kechadi, and Jack Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 130–140.

[12] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, et al. 2024. The Llama 3 Herd of Models. arXiv:2407.21783 [cs.AI] https://arxiv.org/abs/2407.21783

[13] Jahanzeb Maqbool Hashmi, Sourav Chakraborty, Mohammadreza Bayatpour, Hari Subramoni, and Dhabaleswar K. Panda. 2018. Designing Efficient Shared Address Space Reduction Collectives for Multi-/Many-cores. In *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 1020–1029. doi:10.1109/IPDPS.2018.00111

[14] Jiajun Huang, Sheng Di, Xiaodong Yu, Yujia Zhai, Jinyang Liu, Zizhe Jian, Xin Liang, Kai Zhao, Xiaoyi Lu, Zizhong Chen, Franck Cappello, Yanfei Guo, and Rajeev Thakur. 2024. hZCCL: Accelerating Collective Communication with Co-Designed Homomorphic Compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis* (Atlanta, GA, USA) *(SC '24)*. IEEE Press, Article 104, 15 pages. doi:10.1109/SC41406.2024.00110

[15] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and zhifeng Chen. 2019. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc.

[16] Krishna Kandalla, Hari Subramoni, Gopal Santhanaraman, Matthew Koop, and Dhabaleswar K. Panda. 2009. Designing multi-leader-based Allgather algorithms for multi-core clusters. In *2009 IEEE International Symposium on Parallel and Distributed Processing*. 1–8. doi:10.1109/IPDPS.2009.5160896

[17] Pitch Patarasuk and Xin Yuan. 2009. Bandwidth Optimal All-Reduce Algorithms for Clusters of Workstations. *J. Parallel Distrib. Comput.* 69, 2 (feb 2009), 117–124. doi:10.1016/j.jpdc.2008.09.002

[18] Samyam Rajbhandari, Conglong Li, Zhewei Yao, Minjia Zhang, Reza Yazdani Aminabadi, Ammar Ahmad Awan, Jeff Rasley, and Yuxiong He. 2022. DeepSpeed-MoE: Advancing Mixture-of-Experts Inference and Training to Power Next-Generation AI Scale. doi:10.48550/ARXIV.2201.05596

[19] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. ZeRO: Memory Optimizations toward Training Trillion Parameter Models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 20, 16 pages.

[20] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. 2020. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism.* Technical Report. arXiv:1909.08053 [cs.CL]

[21] Siddharth Singh, Prajwal Singhania, Aditya Ranjan, John Kirchenbauer, Jonas Geiping, Yuxin Wen, Neel Jain, Abhimanyu Hans, Manli Shu, Aditya Tomar, Tom Goldstein, and Abhinav Bhatele. 2024. Democratizing AI: Open-source Scalable LLM Training on GPU-based Supercomputers. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC '24)*.

[22] Gemma Team, Morgane Riviere, Shreya Pathak, et al. 2024. Gemma 2: Improving Open Language Models at a Practical Size. arXiv:2408.00118 [cs.CL] https://arxiv.org/abs/2408.00118

[23] Rajeev Thakur and William D. Gropp. 2003. Improving the Performance of Collective Operations in MPICH. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Jack Dongarra, Domenico Laforenza, and Salvatore Orlando (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 257–267.

[24] Truong Thao Nguyen, Mohamed Wahib, and Ryousei Takano. 2018. Hierarchical Distributed-Memory Multi-Leader MPI-Allreduce for Deep Learning Workloads . In *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*. IEEE Computer Society, Los Alamitos, CA, USA, 216–222. doi:10.1109/CANDARW.2018.00048

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *CoRR* abs/1706.03762 (2017). arXiv:1706.03762 http://arxiv.org/abs/1706.03762

[26] Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, et al. 2022. Opt: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068* (2022).

[27] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. 2023. PyTorch FSDP: Experiences on Scaling Fully Sharded Data Parallel. *Proc. VLDB Endow.* 16, 12 (aug 2023), 3848–3860. doi:10.14778/3611540.3611569

[28] Qinghua Zhou, Quentin Anthony, Lang Xu, Aamir Shafi, Mustafa Abduljabbar, Hari Subramoni, and Dhabaleswar K. DK Panda. 2023. Accelerating Distributed Deep Learning Training with Compression Assisted Allgather and Reduce-Scatter Communication. 134-144 pages. doi:10.1109/IPDPS54959.2023.00023