MPI-xCCL: A Portable MPI Library over Collective Communication Libraries for Various Accelerators

Chen-Chun Chen
chen.10252@osu.edu
The Ohio State University
Columbus, Ohio, USA

Kawthar Shafie Khorassani
shafiekhorassani.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Pouya Kousha
kousha.2@osu.edu
The Ohio State University
Columbus, Ohio, USA

Qinghua Zhou
zhou.2595@osu.edu
The Ohio State University
Columbus, Ohio, USA

Jinghan Yao
yao.877@osu.edu
The Ohio State University
Columbus, Ohio, USA

Hari Subramoni
subramoni.1@osu.edu
The Ohio State University
Columbus, Ohio, USA

Dhabaleswar K. Panda
panda@cse.ohio-state.edu
The Ohio State University
Columbus, Ohio, USA

ABSTRACT

The evolution of high-performance computing toward diverse accelerators, including NVIDIA, AMD, Intel GPUs, and Habana Gaudi Accelerators, demands a user-friendly and efficient utilization of these technologies. While both GPU-aware MPI libraries and vendor-specific communication libraries cater to communication requirements, trade-offs emerge based on library selection across various message sizes. Thus, prioritizing usability, we propose MPI-xCCL, a Message Passing Interface-based runtime with cross-accelerator support for efficient, portable, scalable, and optimized communication performance. MPI-xCCL incorporates vendor-specific libraries with GPU-aware MPI runtimes ensuring multi-accelerator compatibility while adhering to MPI standards. The proposed hybrid designs leverage the benefits of MPI and xCCL algorithms and transparently to the end user. We evaluated our designs on various HPC systems using OSU Micro-Benchmarks, and Deep Learning frameworks TensorFlow with Horovod. On NVIDIA-GPU-enabled ThetaGPU, our designs outperformed Open MPI by 4.6x. On emerging Habana Gaudi-based systems, MPI-xCCL was also able to deliver similar performance as vendor-provided communication runtimes.

KEYWORDS

NCCL, RCCL, HCCL, MSCCL, xCCL, GPU, MPI

ACM Reference Format:
Chen-Chun Chen, Kawthar Shafie Khorassani, Pouya Kousha, Qinghua Zhou, Jinghan Yao, Hari Subramoni, and Dhabaleswar K. Panda. 2023. MPI-xCCL: A Portable MPI Library over Collective Communication Libraries for Various Accelerators. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3624062.3624153

SC-W 2023, November 12–17, 2023, Denver, CO, USA

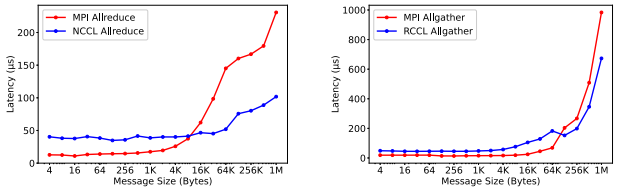
ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3624153

1 INTRODUCTION

The High-Performance Computing (HPC) landscape has significantly evolved over recent years to accommodate a diverse set of accelerators, driven by the growing demand for high computing power in scientific, big data, and deep learning applications. As this demand intensifies, it necessitates the support for various accelerators deployed on HPC systems, particularly in the context of communication runtime for efficient workload scalability on supercomputing systems. Message Passing Interface (MPI) is the defacto communication paradigm used in HPC systems to enable communication across processes, modern GPUs [5, 18, 20], and new network interconnect [17].

To cater to diverse accelerators, the MPI runtime must be extended to integrate the APIs and software designed to harness the capabilities of specific hardware. For instance, NVIDIA GPUs require the MPI runtime to support CUDA-aware communication, while AMD GPUs necessitate ROCm-aware communication runtime. These GPU-aware MPI libraries facilitate direct GPU-to-GPU data transfers, yet their implementation demands substantial domain expertise and extensive effort, often falling short of optimal performance across all message sizes. Therefore, GPU vendors offer vendor-specific communication libraries to attain superior collective communication performance, particularly tailored for deep learning (DL) applications. It inspires the MPI library to harness the capabilities of vendor-specific libraries and incorporate their features into the current designs. As the number of accelerators and vendor-specific APIs expands, the need for a well-defined layer that leverages the underlying APIs and vendor-specific communication support becomes crucial. This layer can be incorporated within the MPI library to establish a unified software framework that effectively exploits the various hardware options available through utilizing specific communication libraries. By streamlining communication and simplifying the integration process, this unified approach enhances the efficiency and adaptability of high-performance computing systems in today's rapidly evolving technological landscape.

Currently, application developers are often responsible for porting or updating their codes to utilize accelerator-specific communication APIs. This requires extensive knowledge and can lead to decreased productivity and potential inconsistencies across various hardware platforms. To mitigate these challenges, a unified



(a) MPI Allreduce vs. NCCL ALlreduce (b) MPI Allgather vs. RCCL Allgather

Figure 1: Comparison of MPI and NCCL Allreduce latency using 32 GPUs (4 nodes) on a DGX A100 system, and MPI and RCCL Allreduce latency using 8 GPUs (4 nodes) on an AMD GPU system. The traditional MPI library is optimal for small message communication, while vendor-provided xCCL is more efficient for large message communication.

communication layer within the MPI library is proposed, which allows developers to focus on optimizing their application codes without mastering accelerator-specific APIs. By implementing this approach, productivity can be increased, and consistent, efficient exploitation of diverse hardware options can be ensured. Ultimately, this contributes to the portability and adaptability of HPC applications in an evolving HPC landscape.

1.1 Motivation

In order to support various DL workloads, many GPU vendors have provided specific support for communication needs through vendor-specific communication libraries such as NCCL [13] (NVIDIA Collective Communication Library) for NVIDIA GPUs and RCCL [2] (ROCm Collective Communication Library) for AMD GPUs. Although these communication libraries are not compliant with the MPI standard and work independently through the particular vendor, they provide an excessive amount of options at the communication layer in addition to GPU-aware MPI libraries from the user's perspective to select from for their communication needs. These libraries are optimized to deliver exceptional communication throughput for DL applications. Lu et al. [21] survey the different vendor-specific communication libraries examine the features and industry use cases of xCCLs, evaluate their performance, and present key takeaways and observations. Consequently, with meticulous fine-tuning by the vendors themselves, they exhibit superior performance for larger message transfers. Figure 1 illustrates a comparison between traditional MPI libraries and vendor-specific libraries, NCCL and RCCL, on NVIDIA and AMD GPU systems. In Figure 1(a), it's evident that the MPI library achieves lower latencies for smaller messages, while NCCL surpasses MPI Allreduce performance beyond the 16 KB threshold. Similarly, on AMD systems in figure 1(b), RCCL initially presents higher overheads up to 64 KB but excels in outperforming MPI Allgather for larger messages. This motivation propels us to seamlessly incorporate vendor-specific libraries into the standard MPI runtime designs, enabling a hybrid communication approach that harnesses the strengths of both for optimal performance across all message sizes.

Integrating vendor-specific libraries into traditional designs is a challenge. With the emergence of new architectures and the evolution of the communication libraries built on top of them, we need to expand and offer a unified communication interface that caters to various vendors/communication libraries, including NVIDIA GPUs, AMD GPUs, and Habana Gaudi accelerators, significantly simplifying the user experience and improving the overall performance.

1.2 Proposed Solution

In this paper, we introduce and evaluate MPI-xCCL (xCCL in short), a unified, portable communication interface that supports various vendor accelerators, enabling users and developers to dynamically leverage the best features from diverse implementations in an application-transparent manner.

From the user perspective, the proposed xCCL offers several advantages: **1)** it allows users to utilize different Collective Communication Libraries (CCLs) across architectures without modifying their code, relying on standard MPI APIs; **2)** it manages the complexities of underlying CCL APIs and logic, such as stream handling for each architecture; **3)** it supports automatic error handling, e.g., falling back to traditional MPI communication if the datatype is not supported in CCL; **4)** it includes common MPI non-blocking collective operations, whereas NCCL only supports five built-in collective operations, requiring users to implement others themselves; **5)** the xCCL design optimizes performance across a wide range of message sizes by selecting the most suitable backend with the hybrid designs; and **6)** it treats CCLs as plug-ins, allowing users to take advantage of the same features and functionalities provided by pure CCLs, even when the underlying CCL version is upgraded.

For developers, xCCL offers **7)** a unified layer for implementing collective operations, eliminating the need to customize algorithms and functions for each new CCL; and **8)** a scalable design that can be easily extended to support upcoming architectures and CCLs, such as oneCCL. By employing xCCL, users can focus on their code without worrying about the underlying details, while developers can streamline the implementation and extension of collective operations across diverse architectures.

MPI-xCCL provides support for communication in a manner that encompasses various vendors, including NVIDIA GPUs, AMD GPUs, and Habana Gaudi accelerators, by exploiting the various vendor-specific supports available combined into one interface referred to as an xCCL MPI Runtime. This refers to an MPI runtime built over the underlying xCCL libraries and APIs, allowing the MPI calls to exploit the underlying vendor-provided communication APIs. We propose hybrid designs that capitalize on the advantages of both MPI and xCCL algorithms across various message sizes. **To the best of our knowledge, this is the first work that has unified support for all different accelerators within MPI. This is also the first work to address communication-level support over the new Habana Gaudi accelerators.**

1.3 Contributions

In this work, we make the following key contributions:

- Propose an interface for GPU-aware MPI libraries over x-Collective Communication Libraries (MPI-xCCL) and APIs (i.e. NCCL [13], RCCL [2], HCCL [9], MSCCL [10], etc.).
- Propose hybrid designs that leverage the benefits of MPI and xCCL algorithms across a range of message sizes.
- Provide a comprehensive performance evaluation of xCCL-based MPI over NVIDIA GPUs using NCCL and MSCCL APIs,

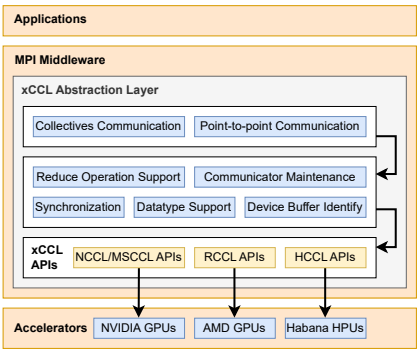


Figure 2: We propose an xCCL Abstraction Layer in MPI that abstracts the common operations in an xCCL runtime. The abstraction design makes it easy to interface with vendor-specific backend implementations while also maintaining MPI standard compliance through being integrated within an MPI communication middleware.

AMD GPUs using RCCL APIs and Habana Gaudi accelerators using HCCL APIs.
- Develop GPU-aware MPI support over Habana Gaudi Accelerators using the HCCL backend.
- Implement Habana Accelerator buffer support for OSU Micro-Benchmarks using Synapse AI Software Suite APIs.

2 BACKGROUND

2.1 Vendor-specific Collective Communication Libraries

As DL models grow and distributed training becomes essential, collective communication in frameworks gains importance. Vendors create communication libraries for their products, like NCCL. NCCL focuses on quick communication between NVIDIA GPUs in dense multi-GPU setups like DGX, especially for DL. On the AMD side, RCCL is a stand-alone for AMD GPUs, dependent on HIP (Heterogeneous Interface for Portability) runtime and ROCm stack. The Habana Collective Communications Library (HCCL) is Habana's implementation of standard routines with an NCCL-compatible API. HCCL targets Habana Gaudi accelerators, built on Gaudi's RoCE V2 RDMA NICs, enabling efficient intra-node and inter-node communication. The Microsoft Collective Communication Library (MSCCL) is an inter-accelerator communication framework for Microsoft Azure, facilitating custom collective communication algorithms across accelerators. MSCCL provides programmability and profiling, enhancing GPU-aware MPI communication. These vendor-specific CCLs enable multi-GPU and multi-node communication, scaling DL training for advanced models.

2.2 GPU-Aware MPI

MPI is a widely used communication scheme that enables communication among distributed processes via messages. However, with exclusive computing devices such as GPUs on modern cluster nodes, directly accessing data residing in device memory becomes a challenge. Traditional MPI processes cannot access GPU buffers directly, and staging data to the host memory for further communication introduces significant overhead.

To address this challenge, CUDA-aware MPI was introduced, which enables the direct transfer of data between GPUs using Unified Virtual Addressing (UVA) and Unified Memory (UM) features. With UVA, the GPU buffer can be directly sent to intra-node peer GPUs or the network adapter without copying data through host memory, bypassing the staging phase and reducing overhead.

3 DESIGN

3.1 Designing an xCCL Abstraction Layer for GPU-aware MPI Library

Starting from NVIDIA, the accelerator vendors have implemented and provided their own libraries to optimize the performance of their devices and intra/inter-node connections. This has resulted in a proliferation of communication APIs that are often specific to each vendor's architecture and programming environment. However, given the success of the NCCL library, other vendors such as AMD, Habana, and Microsoft have proposed similar communication APIs with compatible functionalities. For example, the API of "Reduce" in NCCL is `ncclReduce`, and HCCL changes it to `hcclReduce`. RCCL and MSCCL just use the same `ncclReduce`. Hence, we proposed a unified xCCL abstraction layer to prevent duplicated efforts.

At a lower level, xCCL APIs map corresponding NVIDIA, AMD, Habana, or Microsoft libraries under the `xccl` prefix, offering unified APIs for upper layers. In the high-level implementation, the xCCL abstraction layer empowers MPI developers with a single API to access third-party libraries, sidestepping the need to tailor their algorithms for specific architectures and accelerators. This adaptable approach aggregates existing APIs, extendable to future communication libraries, enhancing productivity. With a unified interface for communication libraries, the xCCL abstraction streamlines development, enabling a focus on optimizing algorithms and applications rather than grappling with compatibility across libraries and hardware platforms.

Figure 2 illustrates the overview of our proposed xCCL Abstraction Layer designs. Besides creating an abstraction layer for xCCL APIs, we can also design and implement the abstraction for helper functions by using the same algorithm and the xCCL runtimes. For example, we use the same algorithm to identify the buffer type. With the xCCL abstraction layer, we do not need to re-implement the functions to fit each different communication library API. At the application layer, an MPI call is made. Within the MPI middleware, the xCCL abstraction layer considers various factors relevant to the scope, i.e., reduction operation support, datatype support, device buffer checks, etc. and then calls the appropriate xCCL API corresponding to the underlying accelerator.

3.2 Built-in Collective Communication Functions

For some computing-involved and common collective communication, such as AllReduce and Broadcast, the vendors often dedicate optimizations and provide specific APIs. For example, `ncclAllReduce` is used by NCCL, RCCL, and MSCCL, and `hcclAllReduce` is used by HCCL. Hence, in our implementation, we just map these vendor APIs to our xCCL APIs and directly call those. In this case, a unified API `xcclAllReduce` is created on top

Listing 1: Pseudo code of xCCL AlltoAllv designs.

```
/* Create XCCL communicator (xccl_comm) */
/* Convert MPI datatype to XCCL datatype (xccl_dt) */
xcclGroupStart();
for (int r = 0; r < comm_size; r++) {
    xccl_ret = xcclSend(((char*)sendbuf) + sdispls[r] * type_size, sendcnts[r],
            xccl_dt, r, xccl_comm, xccl_stream);
    xccl_ret = xcclRecv(((char*)recvbuf) + rdispls[r] * type_size, recvcnts[r],
            xccl_dt, r, xccl_comm, xccl_stream);
}
xcclGroupEnd();
/* XCCL Stream Synchronization */
```

of either `ncclAllReduce` or
`hcclAllReduce` depending on the lower-level system and vendor library. To further enhance the compatibility and functionality of the xCCL Abstraction Layer, we will also implement a checking mechanism for the supported datatype and reduce the operations of the third-party libraries. This is particularly important since more mature libraries, such as NCCL, support a wider range of datatypes, especially those utilized by deep learning applications. On the other hand, HCCL only supports `float` currently. In addition, even NCCL does not fully support all the datatypes that are supported by MPI standards. For example, MPI standards support `MPI_DOUBLE_COMPLEX`, which is widely utilized by FFT applications like heFFTe, but NCCL has no such implementation.

3.3 Customized Send-recv-based Collective Communication Functions

Currently, the CCL APIs only provide 5 collective communications mentioned in subsection 3.2. The other collective calls, such as Gather, are simple send-recv-based communications. There is no vendor-optimized built-in implementation, and users used to have to implement it on their own by utilizing group calls (`ncclGroupStart` and `ncclGroupEnd`) and point-to-point communications (`ncclSend` and `ncclRecv`) to implement other common send-recv-based collective communication functions. To fully support these collective MPI operations, we implemented those functions with our high-level xCCL APIs and provided hooks in MPI runtimes. Users can simply call the standard MPI functions and utilize the operations with a third-party collective communication backend. Listing 1 shows an implementation example of AlltoAllv.

3.4 Hybrid Designs

To further improve the performance and adaptability of the xCCL designs, we introduce hybrid designs that enable users to leverage both vendor-optimized collective communication libraries and the existing MPI implementations. In fact, modern MPI libraries utilize different protocols and algorithms according to different conditions, such as system architectures, MPI operations, and message sizes. Tuning tables are maintained to keep track of the protocols or algorithms that deliver optimal performance under corresponding conditions. With the xCCL Abstraction Layer designs, each operation can be easily encapsulated into one of the MPI algorithms and be called as one of the regular MPI implementations in the tuning tables. In this work, we tune the tuning tables offline, and during runtime, the hybrid designs select the most optimal solution from the tuning tables. In summary, our hybrid design approach improves performance, compatibility, and adaptability for a wide range of communication patterns and hardware configurations.

Table 1: Systems hardware information (single node)

System Component	ThetaGPU(NVIDIA)	MRI(AMD)	Voyager(Habana)
CPU	AMD EPYC 7742	AMD EPYC 7713	Intel Xeon Gold 6336Y
Memory	1TB DDR4	256 GB DDR4	512 GB DDR4
Sockets	2	2	2
Core/sockets	64	64	24
Accelerator/ Node	8 NVIDIA DGX A100 GPUs	2 AMD MI100 GPUs	8 Habana Gaudi Processors
Device Memory/ GPU(HPU)	40GB HBM2	32 GB HDM2	32 GB HDM2

4 EVALUATION

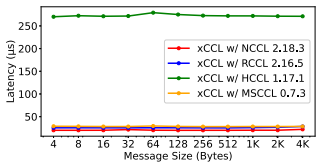
4.1 Experimental Setup

To evaluate our proposed designs over different backends of collective communication libraries, we conducted the evaluations for NVIDIA GPU systems on ThetaGPU at Argonne Leadership Computing Facility (ALCF), AMD GPU systems on an in-house cluster named MRI, and Habana Gaudi Processor systems on Voyager at San Diego Supercomputer Center (SDSC). Table 1 describes the details of a single node of each system. ThetaGPU has comprised 24 NVIDIA DGX A100 nodes. Each node is equipped with 8 NVIDIA A100 Tensor Core GPUs that are connected with the second generation NVIDIA NVSwitch, and is connected with Mellanox ConnectX-6 VPI HDR. MRI is an in-house cluster, and each node is equipped with 2 AMD MI100 GPUs and connected with Mellanox ConnectX-6 HDR. Voyager is designed to support research in science and engineering, especially for artificial intelligence computing. It is famous for being equipped with the Habana Gaudi training and first-generation Habana inference processors and connected with a 400 Gbps interconnect from Arista.

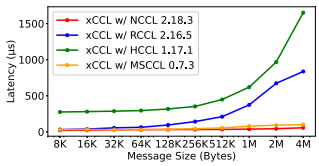
OSU Micro-Benchmarks (OMB) [3] suite supports CUDA and ROCm device buffers, so we can directly use OMB (v7.2) to evaluate our designs on NVIDIA and AMD systems. However, there is no support for Habana device buffer allocation and evaluation, so we implemented a modified OMB suite based on version 7.0.

4.2 Micro-Benchmark Evaluation: Point-to-point

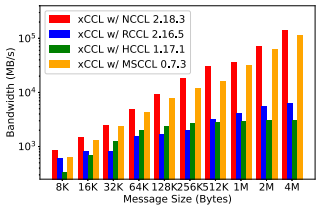
In this section, we assess intra-node and inter-node point-to-point communication. Figure 3 illustrates intra-node latency, bandwidth, and bi-directional bandwidth, aggregating results from 4 distinct xCCL backends. We depict 4 different xCCL backend results in 1 figure, but please note that there is no relationship between the 4 sets of numbers. NCCL exhibits mere 56 µs latency at 4MB, boasting bandwidths of up to 137031 and 181204 MB/s for unidirectional and bi-directional communication. In contrast, RCCL and HCCL backends display latencies of up to 836 and 1651 µs, coupled with bandwidths of 6351 and 3044 MB/s, less than 95% of NCCL's bandwidth. This disparity is attributed to the presence of NVLink on ThetaGPU but PCIe connections for MRI GPUs. On ThetaGPU, MSCCL records a 100 µs latency at 4MB, yielding bandwidth and bi-directional bandwidth of 112439 and 131859 MB/s, respectively. Since MSCCL employs an earlier NCCL version (2.12.12) as a backend, its performance mirrors that of NCCL. Additionally, it's noteworthy that a constant overhead accompanies all 4 backends. The



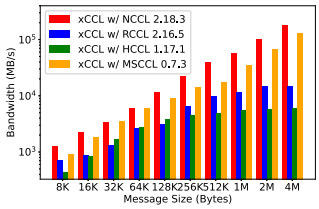
(a) Small Message Point-to-Point Latency



(b) Large Message Point-to-Point Latency

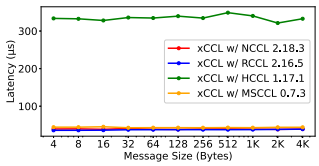


(c) Large Message Point-to-Point Bandwidth (Higher is Better)

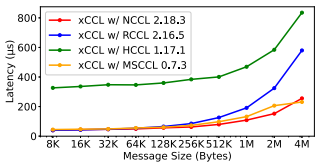


(d) Large Message Point-to-Point Bi-Directional Bandwidth (Higher is Better)

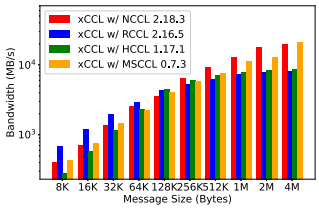
Figure 3: Intra-Node Point-to-Point Performance



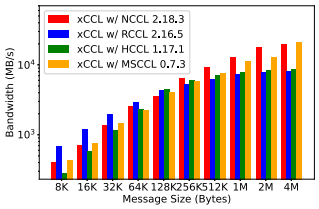
(a) Small Message Point-to-Point Latency



(b) Large Message Point-to-Point Latency



(c) Large Message Point-to-Point Bandwidth (Higher is Better)



(d) Large Message Point-to-Point Bi-Directional Bandwidth (Higher is Better)

Figure 4: Inter-Node Point-to-Point Performance

launch overheads for NCCL, RCCL, HCCL, and MSCCL communications amount to 20, 25, 270, and 28 µs, respectively.

Figure 4 shows the inter-node latency, bandwidth, and bi-directional bandwidth. There is a similar trend compared to the results of intra-node numbers. The overheads of latencies for NCCL, RCCL, HCCL, and MSCCL at 4MB are 255, 579, 835, and 230 µs, respectively.

4.3 Micro-Benchmark Evaluation: Collective

We assess performance using proposed designs employing NCCL, RCCL, HCCL, and MSCCL backends for MPI collective operations, AllReduce, Reduce, Bcast, and AlltoAll, via OMB. Our designs encompass hybrid approaches (labeled as "Proposed Hybrid xCCL") and pure xCCL backends (noted as "Proposed xCCL w/ Pure ..."). We present 1 and 16-node results against pure NCCL and Open MPI + UCX + UCC, and 1 and 2-node results against pure MSCCL and its corresponding NCCL version performance. Pure NCCL and MSCCL outcomes are extracted from OMB NCCL benchmarks (shown as dashed lines). It's important to note the absence of comparable third-party benchmarks for RCCL and HCCL due to the unavailability of corresponding benchmarks in OMB. Nevertheless, we display 1 and 8-node results for RCCL and 1 and 4-node results for HCCL.

Figure 5 displays common collective performance on 1 node. Overall, the performance of proposed pure xCCL designs (blue lines) mirrors that of original vendor-specific xCCL (dotted red lines), highlighting minimal overhead in our implementation. Notably, the proposed hybrid xCCL (red lines) achieves even lower small message latency. For instance, in Figure 5(e), Reduce latencies shrink from 23 to 14 µs for small messages (<8KB). While no RCCL or HCCL benchmarks exist for AMD and Habana architectures, outcomes suggest our designs' adaptability to new architectures. In Figure 5(d), where MSCCL uses NCCL 2.12.12 as the backend, we use pure NCCL 2.12.12 as the baseline. MSCCL outperforms NCCL for medium messages (256B 256KB), and our performance mirrors MSCCL, excelling for small messages (<64B) due to hybrid designs. In NCCL evaluations, we also compared our results to Open MPI + UCX + UCC, revealing our designs' reduced overhead. Notably, our designs excel by 1.1x in Allreduce (figure 5(a)) and a remarkable 2.8x in Alltoall (figure 5(m)) at the 4 KB level.

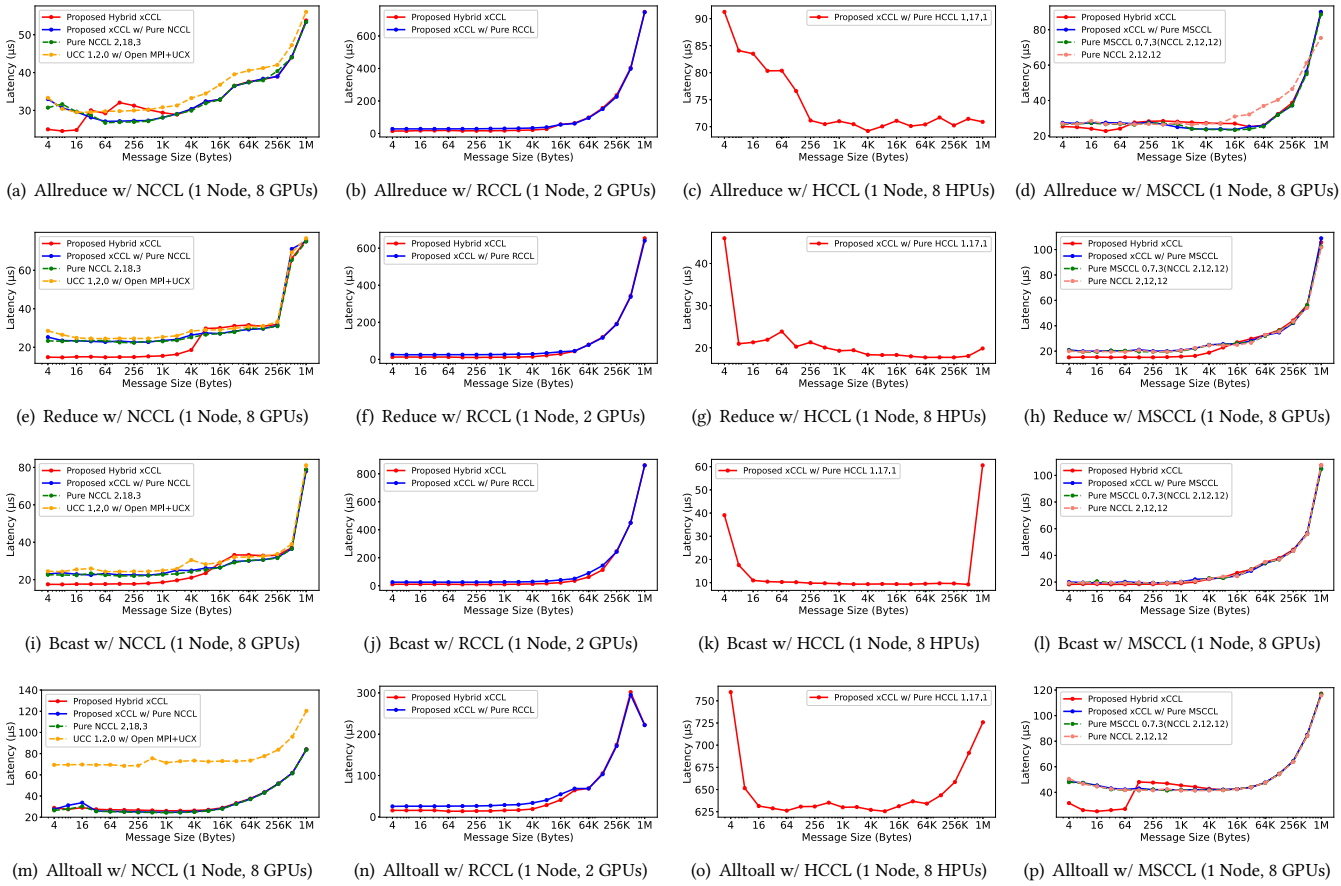
Figure 6 illustrates multi-node common collective performance using NCCL, RCCL, HCCL, and MSCCL. Our observations parallel those of the 1-node figures. Notably, the HCCL backend on Habana systems exhibits more overhead than NCCL, RCCL, and MSCCL backends on NVIDIA and AMD setups. For Allreduce, Reduce, and Bcast, degradations manifest as step curves around 16 and 64 bytes, reaching up to 7x to 12x. In NCCL evaluations, our designs consistently outperform Open MPI + UCX + UCC for nearly all messages, showcasing reduced latencies for small messages, evident in figures 6(a) (Allreduce) and 6(i) (Alltoall).

Comparing our xCCL designs and the pure collective communication libraries provided by the vendors, the curves of xCCL with NCCL/MSCCL and Pure NCCL/MSCCL are almost overlapped, especially the NCCL part. In most cases, there is only ±3% variation between xCCL with NCCL and pure NCCL. These trends demonstrate that there are minimal overheads in our implementations, and the hybrid designs provide even better performance for small messages. Users can readily adopt our xCCL designs with traditional MPI runtimes without experiencing degradation in performance.

4.4 Application-Level Evaluation

This section evaluates the proposed xCCL designs at the application level by DL application TensorFlow with Horovod on all platforms. TensorFlow is a well-known deep-learning framework, and Horovod provides a simple interface for distributed learning.

TensorFlow with Horovod on NVIDIA System: On an NVIDIA system, we assess the performance of our proposed xCCL using two distinct NCCL backends: the latest version and version 2.11.4, harmonizing with TensorFlow and Horovod versions on ThetaGPU. We juxtapose our designs against pure NCCL 2.11.4, Open MPI + UCX, and Open MPI + UCX + UCC. Figure 7 showcases results for batch sizes 32, 64, and 128 on 1 node with 8 GPUs and 16 nodes with 128 GPUs. In Figure 7(a), our xCCL designs (with



(a) Allreduce w/ NCCL (1 Node, 8 GPUs) (b) Allreduce w/ RCCL (1 Node, 2 GPUs) (c) Allreduce w/ HCCL (1 Node, 8 HPUs) (d) Allreduce w/ MSCCL (1 Node, 8 GPUs)

(e) Reduce w/ NCCL (1 Node, 8 GPUs) (f) Reduce w/ RCCL (1 Node, 2 GPUs) (g) Reduce w/ HCCL (1 Node, 8 HPUs) (h) Reduce w/ MSCCL (1 Node, 8 GPUs)

(i) Bcast w/ NCCL (1 Node, 8 GPUs) (j) Bcast w/ RCCL (1 Node, 2 GPUs) (k) Bcast w/ HCCL (1 Node, 8 HPUs) (l) Bcast w/ MSCCL (1 Node, 8 GPUs)

(m) Alltoall w/ NCCL (1 Node, 8 GPUs) (n) Alltoall w/ RCCL (1 Node, 2 GPUs) (o) Alltoall w/ HCCL (1 Node, 8 HPUs) (p) Alltoall w/ MSCCL (1 Node, 8 GPUs)

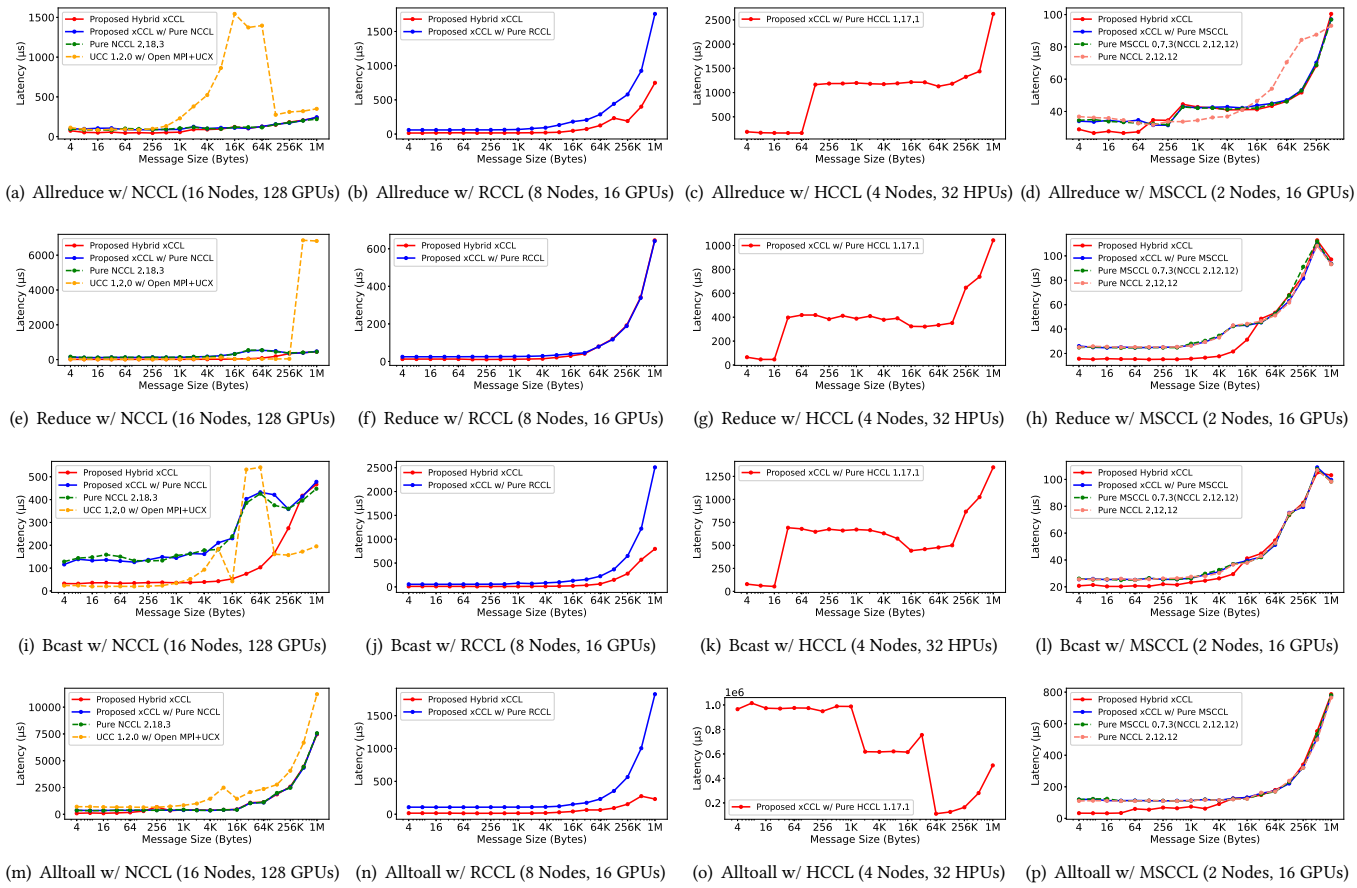
Figure 5: Collective Performance on Single Node (Lower is Better).

NCCL 2.18.3 or 2.11.4) either match or surpass pure NCCL performance. For instance, xCCL achieves 4850 img/sec compared to pure NCCL's 4050 img/sec at batch size 32. Conversely, traditional MPI runtimes—Open MPI + UCX or advanced designs with UCC—yield 3450 or 4480 img/sec at a batch size of 128, 44% or 28% below our designs. Figure 7(b) depicts xCCL's 94600 img/sec throughput with 128 GPUs, 1.35x and 1.5x higher than Open MPI + UCX and UCC with a batch size of 128. Notably, in multi-node evaluations, UCC underperforms Open MPI + UCX by 10%. Figure 10 shows our xCCL's performance with the MSCCL backend, mirroring the NCCL trend, with xCCL achieving 12300 img/sec at batch size 128 on 2 nodes.

TensorFlow with Horovod on AMD System: On an AMD system, we evaluated the performance of our proposed xCCL compared to pure RCCL 2.11.4. The results in figure 8(a) show that our xCCL designs achieve a throughput of 3192 img/sec with a batch size of 64 on 8 AMD GPUs, which is a 25% improvement over pure RCCL. Additionally, figure 8(b) demonstrates a throughput of 7210 img/sec with a batch size of 128 on 16 AMD GPUs, which is a 20% improvement over pure RCCL.

TensorFlow with Horovod on Habana System: We assess the Habana system's performance using the HCCL backend, leveraging prebuilt Habana TensorFlow and Horovod within the container image. In Habana TensorFlow, the communication layer in Horovod is directly implemented via HCCL. To align with this, we adapt the Horovod communication by substituting all `hcclAllreduce` calls with `MPI_Allreduce` operations. Figure 9 showcases performance results on the Habana system for batch sizes 32, 64, and 128. Figure 9(a) illustrates single-node throughput, where xCCL delivers 5139 img/sec throughput with batch size 128, nearly matching pure HCCL's 4936 img/sec. In Figure 9(b), 4 nodes with 32 HPUs each achieve a throughput of 11300 img/sec, with overhead under 1%, for both xCCL and pure HCCL. This evaluation underscores the effortless extension of our xCCL designs to new architectures and collective communication libraries, exhibiting negligible overheads.

This evaluation demonstrates that our xCCL designs yield comparable or superior performance to pure NCCL, RCCL, HCCL, and MSCCL at the application level. Users can enhance performance without altering their implementation; they continue to utilize the familiar MPI runtime. The proposed xCCL framework enables optimal performance for DL and HPC applications using a single MPI library and consistent settings. Notably, as a real-world instance, we encountered errors with pure NCCL 2.18.3 on ThetaGPU and invested substantial effort in testing different versions of TensorFlow, Horovod, CUDA, and NCCL. Eventually, we identified a functional version, 2.11.4. In contrast, our xCCL designs bypass such errors,



(a) Allreduce w/ NCCL (16 Nodes, 128 GPUs) (b) Allreduce w/ RCCL (8 Nodes, 16 GPUs) (c) Allreduce w/ HCCL (4 Nodes, 32 HPUs) (d) Allreduce w/ MSCCL (2 Nodes, 16 GPUs)

(e) Reduce w/ NCCL (16 Nodes, 128 GPUs) (f) Reduce w/ RCCL (8 Nodes, 16 GPUs) (g) Reduce w/ HCCL (4 Nodes, 32 HPUs) (h) Reduce w/ MSCCL (2 Nodes, 16 GPUs)

(i) Bcast w/ NCCL (16 Nodes, 128 GPUs) (j) Bcast w/ RCCL (8 Nodes, 16 GPUs) (k) Bcast w/ HCCL (4 Nodes, 32 HPUs) (l) Bcast w/ MSCCL (2 Nodes, 16 GPUs)

(m) Alltoall w/ NCCL (16 Nodes, 128 GPUs) (n) Alltoall w/ RCCL (8 Nodes, 16 GPUs) (o) Alltoall w/ HCCL (4 Nodes, 32 HPUs) (p) Alltoall w/ MSCCL (2 Nodes, 16 GPUs)

Figure 6: Collective Performance on Multiple Nodes (Lower is Better).

offering easy adaptation by simply adjusting the NCCL backend through the corresponding library path setting.

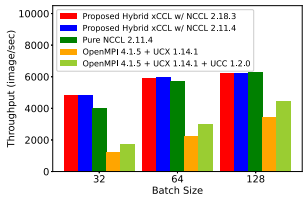
5 RELATED WORK

GPU-aware MPI has emerged as a critical aspect of HPC, with libraries like MVAPICH2 [12] and Open MPI [15] providing support for CUDA and ROCm [1] runtimes, besides, MPICH[11], Intel MPI[8], IBM's Spectrum MPI[7], and Cray MPICH[14] also provide support for accelerators. These libraries have their unique features and capabilities. MVAPICH2 has been optimized for high-performance networks, incorporating the NCCL [13] API and supporting GPUDirect since the early research[20] to transfer data between GPUs in InfiniBand clusters. Since then, multiple optimization strategies[4, 16, 19, 22] have been proposed to significantly accelerate the communication performance of GPU data transfer. Recently added ROCm-aware MPI runtime [18] further extends MVAPICH2 support to AMD GPUs with ROCm. Open MPI is known for its flexibility in supporting multiple networks and fabrics. In the context of vendor-specific CCLs, NCCL [13] focuses on NVIDIA GPUs, while RCCL [2] is designed for AMD GPUs. HCCL [9] is Habana's emulation layer of NCCL, included in the SynapseAI Software library, providing the same collective communication primitives and allowing for point-to-point communication. MSCCL [10] is intended for Microsoft platforms and supports advanced features such as multi-fabric support and RDMA.

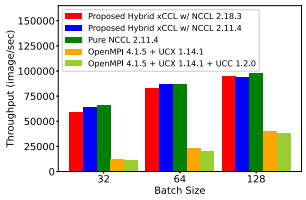
Additionally, the Unified Collective Communication Library [6] (UCC) has been introduced to offer a flexible and feature-rich API for collective communication operations. UCC's design emphasizes scalability, nonblocking operations, flexible resource allocation, and support for hardware collectives. It caters to various HPC, AI/ML, and I/O workloads and supports a wide range of transports, including UCX/UCP, SHARP, CUDA, NCCL [13], and RCCL [2].

6 CONCLUSION

To facilitate optimal communication-level performance for diverse accelerators used in HPC and Deep Learning on supercomputers by different vendors, comprehensive support within communication libraries is crucial. Our paper introduces the xCCL communication runtime, seamlessly integrated with advanced GPU-aware MPI runtimes, providing vendor-specific communication library support. We establish an abstraction layer encompassing NCCL, RCCL, HCCL, and MSCCL APIs. This allows the MPI runtime to dynamically select hardware-specific API calls from various communication libraries across vendors. Our performance evaluation spans ThetaGPU, MRI, and Voyager clusters with NVIDIA GPUs, AMD GPUs, and Habana HPUs, respectively. We comprehensively

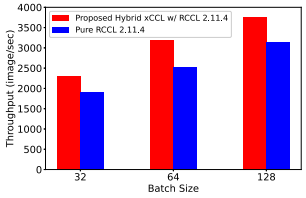


(a) 1 Node (8 GPUs)

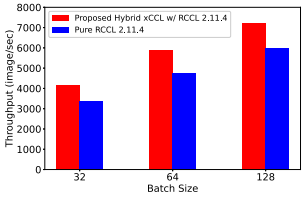


(b) 16 Nodes (128 GPUs)

Figure 7: Performance of TensorFlow with Horovod on NVIDIA System Using NCCL (Higher is Better).

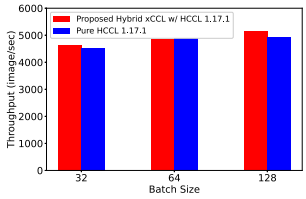


(a) 4 Node (8 GPUs)

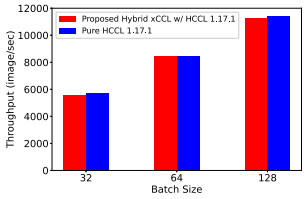


(b) 8 Nodes (16 GPUs)

Figure 8: Performance of TensorFlow with Horovod on AMD System Using RCCL (Higher is Better).

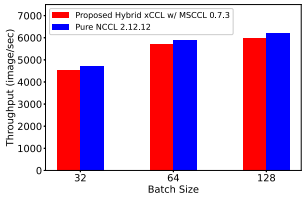


(a) 1 Node (8 HPUs)

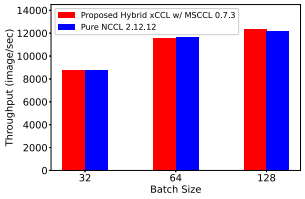


(b) 4 Nodes (32 HPUs)

Figure 9: Performance of TensorFlow with Horovod on Habana System Using HCCL (Higher is Better).



(a) 1 Node (8 GPUs)



(b) 2 Nodes (16 GPUs)

Figure 10: Performance of TensorFlow with Horovod on NVIDIA System Using MSCCL (Higher is Better).

assess intra-node and inter-node communication and collective operations across single and multiple GPU nodes using four communication backends. At the application level, our designs achieved substantial throughput gains over UCC and RCCL by 4.6x and 1.25x. Remarkably, our work pioneers communication-level performance evaluation for upcoming Habana Gaudi Processors. Future work aims to extend support to additional hardware like Intel GPUs or FPGAs and new vendor-specific libraries like oneCCL.

ACKNOWLEDGMENTS

This research is supported in part by NSF #1818253, #1854828, #1931537, #2007991, #2018627, #2311830, #2312927, and XRAC grant #NCR-130002.

REFERENCES

[1] AMD. 2016. Radeon Open Compute Platform. https://rocmdocs.amd.com.
[2] AMD. 2018. RCCL. https://github.com/ROCmSoftwarePlatform/rccl.
[3] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda. 2012. OMB-GPU: A Micro-benchmark Suite for Evaluating MPI Libraries on GPU Clusters. In *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface (EuroMPI)* (Vienna, Austria). 110–120.
[4] Chen-Chun Chen, Kawthar Shafie Khorassani, Quentin G. Anthony, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. 2022. Highly Efficient Alltoall and Alltoallv Communication Algorithms for GPU Systems. In *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. 24–33. https://doi.org/10.1109/IPDPSW55747.2022.00014
[5] Chen-Chun Chen, Kawthar Shafie Khorassani, Goutham Kalikrishna Reddy Kuncham, Rahul Vaidya, Mustafa Abduljabbar, Aamir Shafi, Hari Subramoni, and Dhabaleswar K. Panda. 2023. Implementing and Optimizing a GPU-aware MPI Library for Intel GPUs: Early Experiences. In *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*. 131–140. https://doi.org/10.1109/CCGrid57682.2023.00022
[6] Github. 2023. Unified Collective Communication. https://github.com/openucx/ucc. Accessed: September 26, 2023.
[7] IBM. 2018. IBM Spectrum MPI: Accelerating high-performance application parallelization. https://www.ibm.com/us-en/marketplace/spectrum-mpi.
[8] Intel. 2004. Intel MPI. https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html.
[9] Intel. 2021. HCCL. https://github.com/HabanaAI/hccl_ofi_wrapper.
[10] Microsoft. 2016. MSCCL. https://github.com/microsoft/msccl.
[11] MPICH. 1992. MPICH. https://developer.nvidia.com/nccl.
[12] Network-Based Computing Laboratory. 2001. MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. http://mvapich.cse.ohio-state.edu/.
[13] NVIDIA. 2017. NCCL2. https://developer.nvidia.com/nccl.
[14] OLCF. 2021. HPE CRAY MPI – SPOCK WORKSHOP. https://www.olcf.ornl.gov/wp-content/uploads/2021/04/HPE-Cray-MPIUpdate-nfr-presented.pdf.
[15] Open MPI. 2004. Open MPI: Open Source High Performance Computing. https://www.open-mpi.org/.
[16] Sreeram Potluri, Khaled Hamidouche, Akshay Venkatesh, Devendar Bureddy, and Dhabaleswar K Panda. 2013. Efficient Inter-node MPI Communication Using GPUDirect RDMA for InfiniBand Clusters With NVIDIA GPUs. In *42nd International Conference on Parallel Processing (ICPP), 2013*. IEEE, 80–89.
[17] Kawthar Shafie Khorassani, Chen Chun Chen, Bharath Ramesh, Aamir Shafi, Hari Subramoni, and Dhabaleswar Panda. 2022. High Performance MPI over the Slingshot Interconnect: Early Experiences. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '22)*. Association for Computing Machinery, New York, NY, USA, Article 15, 7 pages. https://doi.org/10.1145/3491418.3530773
[18] Kawthar Shafie Khorassani, Jahanzeb Hashmi, Ching-Hsiang Chu, Chen-Chun Chen, Hari Subramoni, and Dhabaleswar K. Panda. 2021. Designing a ROCm-Aware MPI Library for AMD GPUs: Early Experiences. In *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24 – July 2, 2021, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 118–136. https://doi.org/10.1007/978-3-030-78713-4_7
[19] R. Shi, S. Potluri, K. Hamidouche, J. Perkins, M. Li, D. Rossetti, and D. K. Panda. 2014. Designing Efficient Small Message Transfer Mechanism for Inter-node MPI Communication on InfiniBand GPU Clusters. In *2014 21st International Conference on High Performance Computing (HiPC)*. 1–10.
[20] Hao Wang, Sreeram Potluri, Devendar Bureddy, Carlos Rosales, and Dhabaleswar K. Panda. 2014. GPU-Aware MPI on RDMA-Enabled Clusters: Design, Implementation and Evaluation. *IEEE Transactions on Parallel and Distributed Systems* 25, 10 (2014), 2595–2605. https://doi.org/10.1109/TPDS.2013.222
[21] Adam Weingram, Yuke Li, Hao Qi, Darren Ng, Liuyao Dai, and Xiaoyi Lu. 2023. xCCL: A Survey of Industry-Led Collective Communication Libraries for Deep Learning. *Journal of Computer Science and Technology* 38, 1 (01 Feb 2023), 166–195. https://doi.org/10.1007/s11390-023-2894-6
[22] Q. Zhou, C. Chu, N. S. Kumar, P. Kousha, S. M. Ghazimirsaeed, H. Subramoni, and D. K. Panda. 2021. Designing High-Performance MPI Libraries with On-the-fly Compression for Modern GPU Clusters*. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 444–453. https://doi.org/10.1109/IPDPS49936.2021.00053