Efficient Multi-Path NVLink/PCIe-Aware UCX based Collective Communication for Deep Learning

Yıltan Hassan Temuçin, AmirHossein Sojoodi, Pedram Alizadeh, and Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University
Kingston, ON, Canada
{*yiltan.temucin, amir.sojoodi, pedram.mohammadalizadehbakhtevari, ahmad.afsahi*}*@queensu.ca*

***Abstract*—High-performance communication for very large messages on modern multi-GPU nodes has become increasingly important for Deep Learning workloads. These computing nodes are equipped with state-of-the-art interconnects, such as Nvidia's NVLink and PCIe, to facilitate communications between GPUs, and GPUs with the host processors. In this paper, we take on the challenge to design efficient intra-socket GPU-to-GPU communication using multiple NVLink channels at the UCX and MPI levels, and then utilise it to design an intra-node hierarchical NVLink/PCIe-aware GPU based `MPI_Allreduce` to enhance Horovod + TensorFlow with different models.**

UCX only utilises a small portion of the available NVLink bandwidth for intra-socket GPU-to-GPU communication. We propose a novel data transfer mechanism that stripes the message across multiple intra-socket communication channels and multiple memory regions using multiple GPU streams to utilise all available NVLink paths. Our approach achieves 1.69x and 1.84x higher bandwidth for UCX and Open MPI + UCX, respectively. We observe similar bandwidth improvements for large messages for MPI point-to-point communication when compared to other MPI implementations as they are also limited by data transfers by a single path.

We then propose a 3-stage hierarchical, pipelined `MPI_Allreduce` design that incorporates the new multi-path NVLink data transfer mechanism for intra-socket communications in the first and third stages of the collective, and PCIe and X-bus channels for inter-socket GPU communication in the second stage with minimal interference. For large messages, our proposed algorithm achieves a high speedup when compared to Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL. We also observe significant speedup for the proposed `MPI_Allreduce` for Horovod with TensorFlow with a variety of Deep Learning models.

***Keywords*-MPI, GPU, NVLink, Collective Communication, CUDA, UCX, Deep Learning Workloads**

I. Introduction

Machine Learning and Deep Learning workloads have become the mainstream due to the availability of large datasets and high-end computing systems. However, they put tremendous pressure on the communication subsystem of computing nodes and clusters. As such, their performance is significantly affected by the underlying interconnect and communication runtime.

The usage of GPU accelerators in HPC systems are becoming ever more prevalent. Multi-GPU computing nodes are equipped with sophisticated intra-node interconnects these days to better accommodate the ever increasing demand for high-performance communication between GPUs, and between GPUs and the host processors. For example, Nvidia offers high-speed NVLink interconnects and NVSwitch to aid interconnectivity for their GPUs. Using NVLink over traditional intra-node PCIe interconnects has shown to provide higher performance for data transfer [1], [2], [3].

MPI is one of the most popular programming models for HPC systems. Multiple implementations of MPI exist, including MPICH [4], MVAPICH2 [5], Spectrum MPI, and Open MPI [6]. It supports point-to-point, collective, and remote memory access (RMA) communication operations. MPI collectives, in particular, `MPI_Allreduce` and `MPI_Bcast`, which involve communications among a group of processes, play a crucial role in the performance of MPI applications, including Deep Learning workloads. Collectives are usually implemented on top of point-to-point communication primitives. Therefore, devising efficient point-to-point communication coupled with highly optimised collective algorithms are highly desirable for maximum application performance.

Existing designs use a single NVLink/PCIe communication path to transfer data. In this paper, we take on the challenge to design efficient GPU-to-GPU communication using all available NVLink/PCIe communication paths at the UCX and MPI levels. We will then utilise our point-to-point multi-path scheme to design an intra-node, hierarchical, and pipelined `MPI_Allreduce` collective that uses the NVLink/PCIe interconnects efficiently to significantly enhance Horovod with TensorFlow Deep Learning workloads. The contributions of this paper are as follows:

- We propose a novel multi-path GPU-to-GPU data transfer mechanism that partitions large point-to-point messages across device-to-device and device-to-host/host-to-device channels to utilise all available NVLink paths using UCX one-sided put operation. Our approach achieves 1.69x and 1.84x higher bandwidth for UCX

and Open MPI + UCX, respectively.

- We propose a 3-stage hierarchical, pipelined `MPI_Allreduce` collective design that utilises the new multi-path copy mechanism for intra-socket data transfers, while dynamically selecting NVLink and PCIe channels for different stages of the algorithm to minimise interference. Our experimental results show a speedup of up to 12.25x, 15.63x, 3.72x, 1.48x, and 1.38x against Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL, respectively.
- We evaluate the impact of the proposed multi-path copy and `MPI_Allreduce` design at the application layer. For Horovod with TensorFlow and VGG16, we observe up to 2.98x, 3.42x, 3.22x, 1.23x, and 3.24x speedup over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL, respectively. For ResNet50, we achieve 1.50x, 1.57x, 1.22x, 1.24x, 1.23x speedup over these MPI libraries, respectively.

The rest of this paper is structured as follows: In Section II, we present relevant background information about UCX and Open MPI. In Section III, we discuss our motivation for the proposed work. Section IV discusses the research related to this paper. We present our proposed point-to-point and collective design along with its implementation in detail in Section V and we evaluate the performance of our design in Section VI. Finally, Section VII concludes the paper and comments on future directions.

II. Background

A. UCX

UCX (Unified Communication X) is an RDMA-based point-to-point communication library for modern low latency, high bandwidth interconnects [7]. It provides an abstract interface for communication that allows for network acceleration across many interconnects. In Figure 1, we see a simplified diagram of UCX relevant to this paper.

1) UCP Layer: The UCP layer of UCX implements high level protocols which are used by other communication libraries such as MPI. UCP supports Remote Memory Access (RMA), active messages, and tag-matching operations, among others. The tag-matching interface is the most relevant to our work as it supports the send-recv semantics of MPI. For this interface, Open MPI implements both the eager and rendezvous protocols. The UCP layer uses the UCT layer to implement these different protocols over a wide range of transports.

2) UCT Layer: The UCT layer is a transport layer that abstracts the data movement across different memory regions. This layer uses low-level APIs such as InfiniBand Verbs, libfabrics, GDRCopy, and CUDA IPC to allow for efficient access to hardware with minimal overhead. This layer defines interfaces for small messages (short), buffered copy-and-send (bcopy), and zero-copy (zcopy) operations. As the focus of this paper is on large GPU messages, we will use the zcopy operation for the CUDA IPC component of UCT layer. This component handles intra-node GPU-to-GPU communication semantics via the usage of CUDA IPC. First, the receiver process places the CUDA IPC memory handle into shared memory. Then, the sender opens the handle and uses a Put operation to place the data into the remote process.

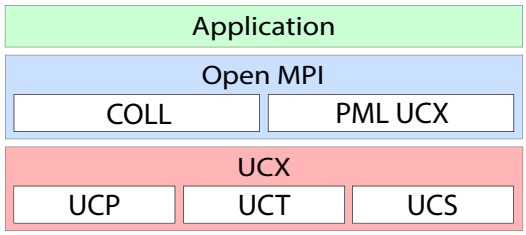


Figure 1: Simplified Software Stack of Open MPI + UCX

B. Open MPI

The Message Passing Interface (MPI) [8] is the most popular programming model used in high-performance computing. In MPI, communication is realised by explicit movement of data from the address space of one process to another. Accordingly, MPI provides support for various types of communications such as point-to-point, collective, and one-sided. GPU support has been added to the well-known implementations of MPI such as MPICH [4], MVAPICH2 [5], and Open MPI [6]. GPU support may follow a general approach that involves staging the GPU data into the host buffer and leveraging the CPU-based MPI routines. However, it may provide GPU-awareness that enables direct communications between GPU buffers.

For this paper, we are using Open MPI with UCX for the PML layer. Open MPI can use other point-to-point components in the PML, such as `ob1`, but we observed better performance when using UCX on our platform. When using point-to-point communication in MPI, we are directly using the UCP interface of UCX. For collective communication, UCX is abstracted through the PML layer, as shown in Figure 1. Open MPI supports various flat algorithms for `MPI_Allreduce`, where the algorithm selection is based on process count, message size, and other system-related information. Such algorithms include ring, segmented ring, reduce-scatter-allgather, recursive doubling, nonoverlapping, and linear. For `MPI_Allreduce` and resident data in GPU global memory, Open MPI uses a CPU-based allreduce.

III. Motivation

Research has shown the performance of GPU based Deep Learning workloads, such as Horovod + TensorFlow, are highly dependent on the performance of `MPI_Allreduce` collective communication operations that use very large

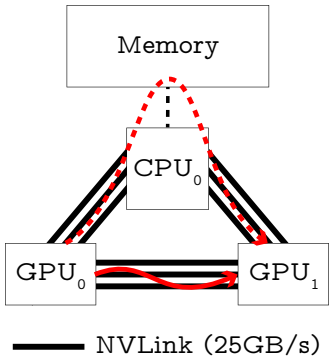


Figure 2: A single socket of an IBM AC922 (8335-GTH) shown with the multi-path copy in red. Host stage copy is dotted and P2P is in solid colour.

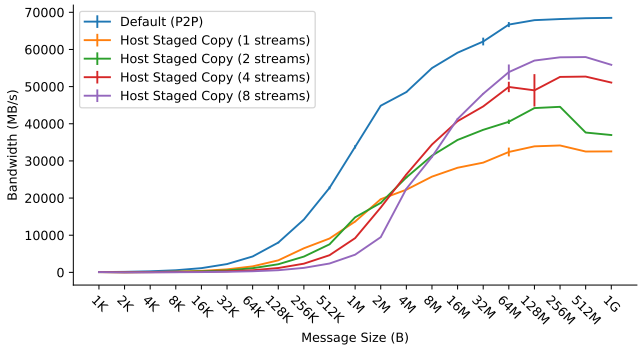


Figure 3: P2P copies compared to host-staged copies with different number of streams.

messages [9], [10]. One way to achieve performance is to use collective algorithms that target large messages and allow overlap and pipelining to occur. In addition, hardware/software resources such as in-GPU reduction, high-performance intra-node interconnects, and efficient data transfer mechanisms must be properly utilised. This research aims at using such measures to achieve high performance for intra-node `MPI_Allreduce` operations. As collectives are mostly designed on top of point-to-point communication, we show that there are opportunities for efficient utilisation of all available communication channels/paths in modern multi-GPU nodes.

Looking over Figure 2, for a single socket of a contemporary multi-GPU node, one can realise that each GPU can be reached through two distinct sets of NVLink communication channels: one connecting it to the host and the other to the adjacent intra-socket GPU. Current MPI implementations send data directly from GPU_0 to GPU_1 via the NVLink connecting the two devices for point-to-point communication. This results in the NVLinks connected to the host to be idle during the data transfer. This observation motivated us to utilise all available intra-socket paths to increase the point-to-point communication bandwidth between the two GPUs.

We investigated the bandwidth we could achieve when transferring the data through the host alone. We copied data from GPU_0 to GPU_1 via CUDA registered host memory. We also split the data into chunks and placed those chunks on GPU streams. For example, when using four streams we partitioned the buffer into four chunks and assigned one stream per chunk. Figure 3 shows the UCX Put bandwidth with `cuda_ipc` as the transport and `zcopy` as the sender-side data layout using `ucx_perftest`. We can observe that an increasing amount of bandwidth (up to 53GB/s) for intra-node GPU-to-GPU data transfer is wasted by not using the host-staged copy alongside the peer-to-peer (P2P) data copy. This motivated us to investigate message striping and partitioning the message across the two P2P and host-stage copy channels to arrive at a much higher bandwidth than the direct P2P copy can provide. At this stage, we speculated that we might be able to achieve around 125GB/s bandwidth between the two devices. This is significantly more than the 72GB/s we were able to measure when using the P2P copy alone. The full design for point-to-point communication will be explained in detail in Section V. We will then propose a hierarchical intra-node all-reduce algorithm that could effectively combine the proposed point-to-point approach alongside X-bus and PCIe interconnects to achieve a high-performance `MPI_Allreduce` operation.

IV. Related Work

There is an abundance of research regarding collective communication operations, but we will focus only on the `MPI_Allreduce` collective in this work. The Reduce-Scatter-AllGather (RSA) algorithm presented in [11] tries to optimise bandwidth for any buffer size or process count. Bandwidth optimised approaches have been designed for the broadcast and reduce stages of an allreduce collective in which two binary trees span all processes [12]. This approach allows for collectives to achieve nearly twice the bandwidth of other algorithms. In [13], Thakur et al. investigate the flat algorithms in MPICH, including ring, recursive doubling, and RSA for reduce and allreduce operations. They observed that the ring algorithm performed better for large message sizes.

In recent years, workloads have begun to rely much more on GPU accelerators. In this regard, proposals to extend MPI to GPUs have appeared in literature [14]. NVIDIA introduced Inter-Process Communication (IPC) with CUDA 4.1. This is a fairly old CUDA feature, but today it is still one of the prominent methods of transferring data between GPU buffers allocated within different address spaces on the same compute node. In [15], Potluri et al. explored using CUDA IPC features within MVAPICH2-GDR. They saw up to 74% latency improvement for 4MB messages compared

to host-staged data transfer between GPUs. Today, most MPI libraries provide CUDA-Aware MPI communication, and they often use CUDA IPC to enhance performance on GPU workloads.

Faraji and Afsahi [16] proposed a GPU shared-buffer aware `MPI_Allreduce` collective design, where MPI processes used CUDA IPC to transfer their GPU-resident data to a common shared-buffer area inside GPU global memory for in-GPU reduction and accelerating the collective. In [17], the same authors, for a set of communications within a collective for each process, used a combination of the host-staged and CUDA IPC copies for inter-process communications. By selecting the correct number and type of the copies, their algorithms were capable of efficiently leveraging the MPS and Hyper-Q feature and provided enhancement for medium and large messages. In [10], Chu et al. used a combination of host-staged copies along with GPU global memory to allow for the acceleration of `MPI_Allreduce` and Deep Learning workloads. However, the work in [17] and [10] are different than our proposed point-to-point approach in this paper, where we use multiple paths, host-staged, and peer-to-peer copy to transfer a *single* message that is stripped into multiple chunks over multiple GPU streams to improve the communication performance for large messages. While there are many works on virtual/physical multi-rail, channel bonding, and message stripping for high-speed networks, such as InfiniBand [18] and Ethernet, to the best of our knowledge, this is the first study to understand the impact of utilising all communication channels and message striping in multi-GPU nodes to enhance point-to-point communication performance, the collective algorithm on top of it, and the Deep Learning workloads.

There has been a great depth of research into hierarchical algorithms for clusters [19], [20], [21], [22]. There has been work into the impact of hierarchy on derived data types [21], CPU cache hierarchy [19], PAP-aware algorithms [20] and using Mellanox multi-connection features [22].

For applications in Deep Learning. we are mostly interested in GPU specific hierarchical collectives. Research in this area is not new. Chu et al. [23] showed that GPU-based reduction was more performant than host-based reduction for large messages in large clusters. Faraji and Afsahi [24] studied various GPU-aware collective algorithms at each level of hierarchy, from intra-GPU to intra-node inter-GPU, and inter-node GPU data transfer.

With the emergence of modern high-bandwidth interconnects such as NVLink, research has focused on their impact on communication performance. In [1], Pearson et al. evaluated the characteristics of CUDA communication primitives on high-bandwidth interconnects to understand memory transfer behaviour across different memory regions. Tallent et al. presented the impact of NVLink and PCIe interconnects on Deep Learning workloads [2]. In [3], Li et al. evaluated such interconnects with a multi-GPU benchmark suite.

In [25], Awan et al. focused on `MPI_Allreduce` and moved the computational part of the collective to GPUs, resulting in significant improvements in Horovod's synthetic benchmarks. Alongside kernel-based reduction, Chu et al. studied the underutilisation of NVLinks [10] by taking the physical topology of the system into account. Our proposals differs from these works as we target the underutilisation NVLinks at the point-to-point layer and that our collective design leverages multiple possible communication during its execution.

V. Design and Implementation

In this section, we propose a new mechanism for intra-socket GPU-GPU data transfers and then use it to propose a new `MPI_Allreduce` design.

A. The Proposed Multi-Path Copy Design

For data transfers between GPU_0 and GPU_1, data would usually be sent via the NVLink directly connecting the two devices. However, Figure 2 shows the two paths in which we can send data from GPU_0 to GPU_1. In our design, we split the send buffer into two and transfer each data segment via an independent data channel, one through NVLinks directly connecting the two devices, and the other through the host-staged copy. As discussed in Section II, point-to-point communication in Open MPI + UCX is implemented with RMA put operations inside UCX. We placed our code into the `cuda_ipc` component of the UCT layer of UCX.

Sending data between GPUs via the host requires staging in host memory. We used the CUDA Driver API call `cuMemAllocHost()` to allocate a region of host memory that is page-locked and accessible to the GPU to allow for read and writes with a much higher bandwidth. We split the data transfers to host memory into multiple chunks that are placed on individual streams to allow for pipelining between D2H and H2D copies. For our study, we experimented with 1, 2, 4, and 8 chunks. The number of chunks is different for each message size and is based on the peak achieved bandwidth results shown in Figure 3. As we use a mixture of host-staged and D2D copies, the ratio in which we split the send buffer will have an impact on aggregate bandwidth. We roughly send around 25-30% of the message via the host and the remainder directly to the adjacent GPU. A different volume of data is sent for each path to account for the bandwidth mismatch. We record the optimal number of chunks and message distribution for each message size in a static tuning table. Although the CUDA IPC code was already implemented within UCX, we made a small change in how the IPC handles were opened. We first set the device context to the remote GPU, the destination of our PUT operation. We then open the IPC handle, and finally set the device context back to the original context. This is not

Algorithm 1: Multi-Path Copy Algorithm

Input: sbuf, host_buf, data_size, host_share, n_host_streams
Output: dbuf

```
1  host_dsize = data_size * host_share;
2  host_chunk_dsize = host_dsize / n_host_streams;
3  d2d_dsize = data_size - host_dsize;
4  do in parallel
5  |  Copy d2d_dsize bytes from sbuf to dbuf;
6  |  for i ← 0 to n_host_streams by 1 do in parallel
7  |  |  Copy host_chunk_size bytes from sbuf to
   |  |    host_buf[i];
8  |  |  Wait for data in host_buf[i];
9  |  |  Copy host_chunk_size bytes from host_buf[i]
   |  |    to dbuf;
10 |  end
11 end
```

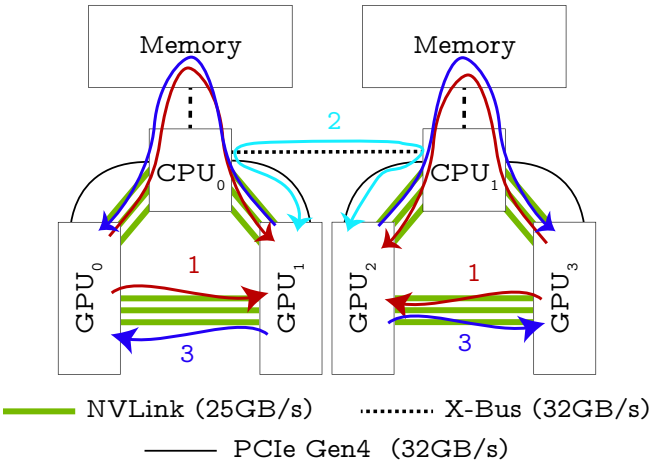


Figure 4: Physical topology of IBM AC922 (8335-GTH) machine shown with the 3 steps of the proposed collective algorithm.

required for peer-to-peer copy, but this was needed to ensure our host-staged copies were pipelined.

Algorithm 1 shows the final stage of our design that combines the host-staged data transfer with the device-to-device copy. In Line 1 to 3, we calculate the data sizes which we must send via the host and also directly to the other GPU. Their shares are predetermined from experiments for each message size, based on the maximum aggregate bandwidth achieved. Once we determine the size of data that we are sending through the host, we copy the data chunks to the host memory on individual CUDA streams, and then from the host memory to the destination GPU when the data chunk is available in the host memory in its entirety (Lines 7 to 9). In parallel, we copy the data from the source GPU directly to the destination GPU (Line 5).

B. The Proposed `MPI_Allreduce` Collective Designs

In this section, we explore two designs for `MPI_Allreduce`, with the goal of impacting Deep Learning workloads, such as Horovod. Therefore, we focus on large GPU messages using `MPI_IN_PLACE` with the `MPI_Op` `MPI_SUM`. Horovod uses other `MPI_Datatype` values and `MPI_Op` operations for CPU and GPU messages, but their frequency totals less than 1% for GPU messages. Thus, we are targeting what we believe is the major bottleneck in the application.

1) `MPI_Allreduce` with GPU Kernel Reduction: As stated earlier, Open MPI uses a CPU based allreduce even when the data is resident in the GPU global memory. This approach translates to poor performance due to D2H and H2D copies for large messages. Also, the NVLinks directly connected between the two GPUs are not used.

We modified the existing Open MPI implementation of `MPI_Allreduce` to use NVLinks between the two intra-socket GPUs, as well as GPU buffers for its temporary buffers for staging rendezvous size messages. These GPU memory temporary buffers allows UCX to send directly between GPUs. Changing these buffers allows us to use the CUDA-Aware features of UCX and CUDA IPC transport layer, and that is where we implemented the multi-path copy mechanism in Section V-A. As our new design no longer copies data to the host, we use a simple CUDA kernel for reduction. As per the default Open MPI implementation, we use the Recursive Doubling algorithm for up to 16KB messages, the Ring algorithm for 16KB to 4MB messages, and the Segmented Ring algorithm for larger than 4MB messages, respectively.

2) The Proposed Hierarchical `MPI_Allreduce` with Multi-Path Copy: There is a large performance gap between intra- and inter-socket bidirectional bandwidth, around 130GB/s compared to 56GB/s. Our objective is to minimise the inter-socket data transfer and use the multi-path copy mechanism, presented in Section V-A, in our collective. Therefore, we propose a hierarchical, pipelined GPU based algorithm with kernel reduction for `MPI_Allreduce` that is designed to use the multi-path copy mechanism for intra-socket communications in Step 1 and Step 3 of the algorithm. Figure 4 shows the three steps of the proposed algorithm. This algorithm uses the same GPU buffers and GPU kernel reduction as in the algorithm presented in Section V-B1. In its simplest form, the proposed algorithm has three steps:

1) GPU_0 and GPU_3 send their data to GPU_1 and GPU_2 (the intra-socket leaders), respectively, to reduce their data.
2) The leaders, GPU_1 and GPU_2, exchange and reduce their data.
3) The leaders GPU_1 and GPU_2 send the reduced data back to GPU_0 and GPU_3, respectively.

Due to the large intra/inter-socket bandwidth discrepancy, Step 2 of this algorithm takes roughly two to three times longer than Step 1 or Step 3. Thus, it would be beneficial to overlap data transfer between Step 1 and Step 2, and between Step 2 and Step 3. To achieve this overlap we use pipelining by chunking the send buffer into multiple chunks and having each chunk execute the algorithm independently. The chunking in the algorithm is in addition to the chunking we described earlier in regards to the multi-path copy.

Design Optimisations: In [3], [1], the authors noted that using `cudaDeviceDisablePeerAccess()` and `cudaDeviceEnablePeerAccess()` allows switching between PCIe and NVLink when both interconnects exist between two GPUs on a platform. Given the fact that between NVLink and X-Bus, the bottleneck is the X-Bus, and that the performance of PCIe Gen4 is the same as X-Bus, for inter-socket Step 2, rather than using the NVLink-X-Bus-NVLink path for communication between GPU 1 and GPU 2, we use the PCIe-X-Bus-PCIe path for certain message sizes. We observed that for each message size, different inter-socket paths would yield the lowest latency for our collective. Therefore, we dynamically switch between the two at runtime. We also noticed that our multi-path copy mechanism has some minor interference issues with inter-socket communication. To minimise this, we dynamically use our multi-path copy or the P2P copy if there are still any messages in transit between the two sockets.

VI. Performance Evaluation and Analysis

Evaluation at each software layer is important to understand the impact of our design. Therefore, in this section, we present the results for UCX, MPI point-to-point and collective communication, and finally with Horovod.

A. Experimental Setup

Experiments were conducted on the Mist compute cluster at the SciNet HPC Consortium. Mist is an IBM POWER9 AC922 machine with two sockets for a total of 32 cores and 382GB of memory. Each node has four Nvidia V100 (32GB) GPUs per node, with three NVLinks between intra-socket GPUs and to the host processors, as shown in Figure 4. Mist uses the GNU/Linux distribution REHL 7.6. For our studies, we have used Open MPI 4.0.4rc2 with UCX 1.8.0, Open MPI + HPC-X (with UCX and HCOLL) from HPC-X v2.7, Spectrum-MPI 10.3.1, MVAPICH2-GDR 2.3.5, NCCL 2.5.6, and Horovod 0.20.3 with TensorFlow 1.15.2. For our application studies with HPC-X, we used Horovod 0.19.2 as we had runtime issues with Horovod 0.20.3.

B. Micro-benchmark Results

1) UCX Put: As MPI point-to-point communication operations are implemented on top of UCX, we first show the UCX zero-copy Put results for GPU-to-GPU data transfer. Figure 3 shows that varying the number of streams has an impact on bandwidth performance, and that we can achieve a lower bandwidth when using host-staged copies. Therefore, when partitioning the send buffer we send a smaller percentage of data via the host, as discussed in Section V-A. Optimal values were found for each message size. We use between 1 to 6 streams and roughly send 25-30% of the message via the host for optimal results. We then placed this tuning table in the UCX library. Figure 5 shows the multi-path copy results for intra-socket data transfers. We can see the peak bandwidth increase from around 72GB/s to 120GB/s (1.67x) when using the proposed multi-path copy mechanism.

2) MPI Point-To-Point: After observing significant performance improvement at the UCX layer for large messages, the investigation was extended to the MPI layer. For our MPI tests, we use the Ohio State University Micro-Benchmark (OMB) suite [26] to gather the uni-directional and bi-directional bandwidth results.

Figure 6 shows the uni-directional bandwidth results with a window size of 1 and 64 between two GPUs on the same socket. The default window size of the `ucx_perftest` is 1 whereas it is 64 for OMB. Modifying OMB's window size allows us to directly correlate the performance we see for the Put operation with MPI point-to-point communication. We observe very similar results to the `ucx_perftest` with a window size of 1. This shows that the performance seen at the UCX layer has a direct impact on MPI performance as we can achieve close to 122GB/s (1.69x) bandwidth. Increasing the window size to 64, we obtain a slightly higher bandwidth measurement of 134GB/s (1.84x). This is clear as larger window sizes can better saturate the NVLinks.

We extend our study to MPI bi-directional bandwidth test, as shown in Figure 7. For a window size of 1 and 64, we get a peak bandwidth of 189GB/s (1.38x) and 182GB/s (1.33x), respectively. We see a slightly lower bandwidth for a window size of 64, and this starts after 128MB message size. We are currently investigating the reason behind this, but we suspect that for larger window sizes each stream has more work to do, causing a slight synchronisation issue.

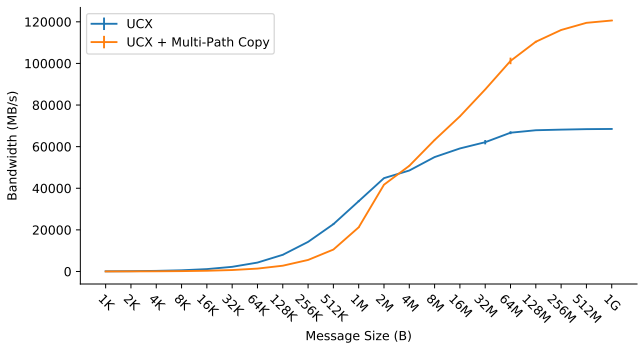
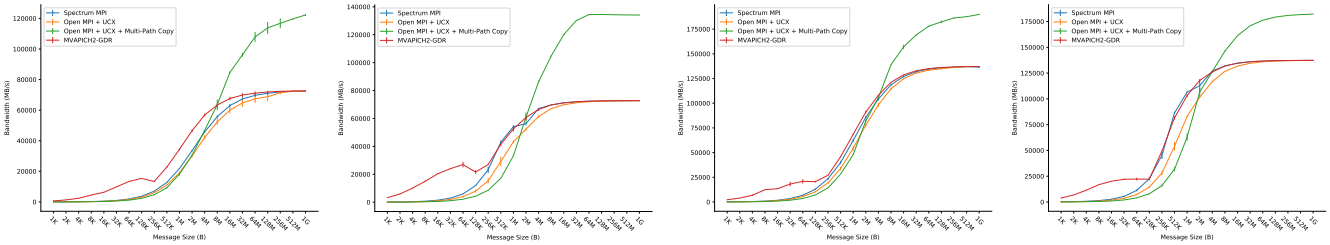


Figure 5: P2P copies compared to the multi-path copy.

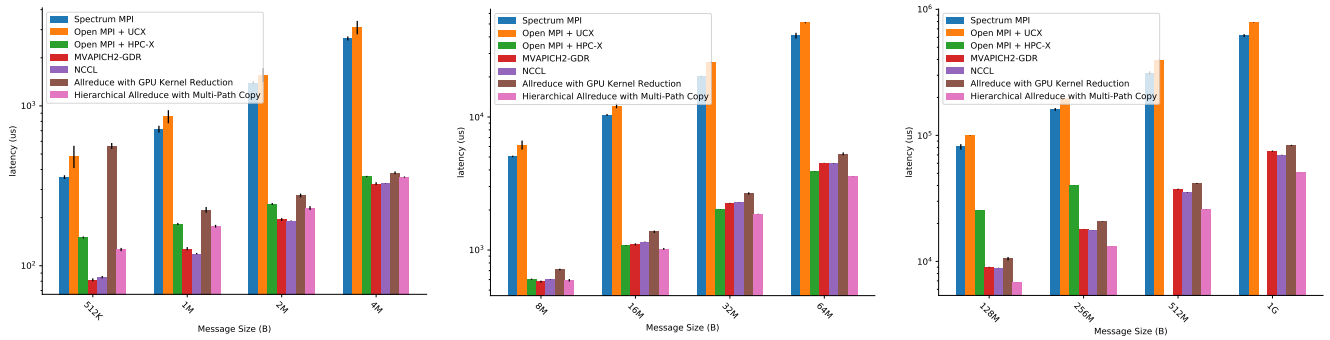


(a) Window Size 1 (b) Window Size 64

Figure 6: MPI point-to-point uni-directional bandwidth with window sizes of 1 and 64.

(a) Window Size 1 (b) Window Size 64

Figure 7: MPI point-to-point bi-directional bandwidth with window sizes of 1 and 64.



(a) Medium to Large (b) Large (c) Very Large

Figure 8: `MPI_Allreduce` results comparing our proposed hierarchical allreduce with multi-path copy as well as the allreduce design with GPU kernel reduction against Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL

From the uni-directional and bi-directional results, it is evident that the multi-path approach outperforms Spectrum MPI, Open MPI, and MVAPICH2-GDR by quite a large margin at large messages, starting at 2MB. This improvement is highly significant for Deep Learning or HPC applications that use very large messages.

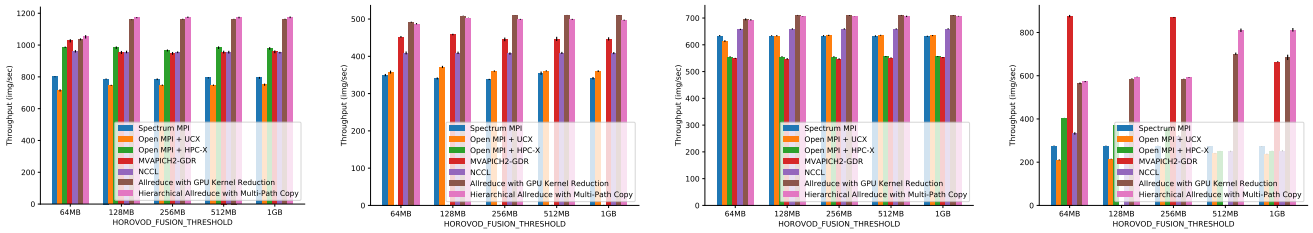
3) `MPI_Allreduce`: For our `MPI_Allreduce` test, we have configured data to be allocated on the GPU for the receive buffer. For the send buffer, we have modified the OMB benchmark so that it uses `MPI_IN_PLACE` to mimic Horovod's MPI API calls. For our comparison with NCCL, we have used `nccl-tests`.

Figure 8 presents the micro-benchmark results for `MPI_Allreduce` for message sizes in 512KB-1GB range, as our focus is on large messages in this paper. For all tests, we have used four processes on a single node with each bound to a single GPU.

Open MPI + UCX has the highest latency for all message sizes, with Spectrum MPI narrowly outperforming it. We start to see the benefit of our `MPI_Allreduce` algorithm with GPU kernel reduction over Open MPI + UCX and Spectrum MPI after 1MB messages. For large messages, the performance improvement is significant. We see a 3.8x and 9.5x speedup over Open MPI + UCX for 1MB and 1GB messages, respectively. We believe that both the idle NVLinks and GPU kernel reduction have contributed to this performance gain in some capacity. This algorithm though still falls short in comparison to NCCL.

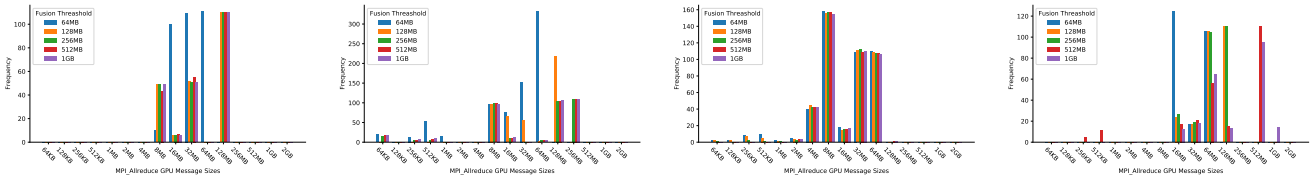
Our proposed hierarchical allreduce with multi-path copy performs better than our allreduce with GPU kernel reduction for all message sizes. It also outperforms Open MPI + HPC-X for all message sizes and NCCL for message sizes greater than 8MB. The proposed algorithm outperforms Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL with a speedup of 11.47x, 14.33x, 1.09x, 1.26x, 1.25x, respectively, for 64MB messages. We outperform Spectrum MPI, Open MPI + UCX, MVAPICH2-GDR, and NCCL at 1GB messages by 12.25x, 15.63x, 1.47x, and 1.38x, respectively. Results for Open MPI + HPC-X at 512MB and 1GB are not present as we faced CUDA 'out of memory' errors.

Our design shows significant performance improvements over Open MPI + UCX and Spectrum MPI. We suspect that this is because `MPI_Allreduce` in these libraries use a CPU based reduction even for GPU resident data. Our design also outperforms Open MPI + HPC-X, NCCL and MVAPICH2-GDR, which are all GPU-optimised communication libraries.



(a) ResNet50 (b) ResNet152 (c) DenseNet201 (d) VGG16

Figure 9: Horovod + TensorFlow throughput with different models and different values of `HOROVOD_FUSION_THRESHOLD`. A batch of 32 is used.



(a) ResNet50 (b) ResNet152 (c) DenseNet201 (d) VGG16

Figure 10: Message sizes used by `MPI_Allreduce` for Horovod + TensorFlow with different models, and `HOROVOD_FUSION_THRESHOLD`. A batch size of 32 is used.

C. Application Results

Figure 9 presents the performance of our proposed design with Horovod with TensorFlow and four Deep Learning models: ResNet50, ResNet152, DenseNet201, and VGG16. We varied the Horovod tuning parameter `HOROVOD_FUSION_THRESHOLD` to see if any additional performance could be gained for the proposed MPI designs [27]. The default value of this parameter is 64MB.

For ResNet50, we see a throughput speedup of up to 1.49x, 1.56x, 1.19x, 1.23x, and 1.21x over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL, respectively. With a fusion threshold larger than 128MB we see minimal change in our performance. With ResNet152 and for a fusion threshold of 64MB, we observe a throughput speedup of 1.40x, 1.36x, 1.08x, and 1.19x over Spectrum MPI, Open MPI + UCX, MVAPICH2-GDR, and NCCL, respectively. Our design outperforms other implementations, but we noticed for this model, there was a slight performance drop using our proposed hierarchical algorithm compared to our allreduce with GPU kernel reduction. For ResNet152, we were unable to obtain results for Open MPI + HPC-X as this model would cause the application to segfault. A modest performance speedup of 1.09x, 1.13x, 1.25x, 1.26x, and 1.06x is seen over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL with DenseNet201, respectively.

VGG16 shows the highest performance improvement when using our proposed collective. We observe a speedup of 2.08x, 2.72x, 1.84x, and 1.72x over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, and NCCL for 64MB buffers, respectively. With this model, we saw that tuning the framework improved the performance further. This tuning gave us 2.98x, 3.42x, 3.20x, and 3.21x speedup for 1GB buffers over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, and NCCL, respectively. For VGG16, we present the results for MVAPICH2-GDR for completeness, as MVAPICH2-GDR would often hang and not return results. We had a 5% success rate for job submissions for this model. This is why only 3 out of 5 data points are presented in this figure. That said, for a buffer size of 64MB we observe a speedup of 0.67x and 1.23x at 1GB. Overall, it is clear from these results that the performance improvement for Horovod + TensorFlow with the proposed `MPI_Allreduce` is significant, but fairly dependent on the model.

We profiled `MPI_Allreduce` during Horovod's execution for different Deep Learning models to draw a connection between our proposed design and the application performance. Figure 10 presents the frequency of message sizes used by `MPI_Allreduce`. With ResNet50, increasing the fusion buffer from 64MB to 128MB changes the message sizes. We see a reduction in messages in 16-64MB range but an increase for 128MB. As we continue to increase the threshold we see no further changes. When comparing this observation to the throughput results in Figure 9, we see that throughput increases for our designs when the threshold changes from 64MB to 128MB but stays constant afterwards. For ResNet152 we observe similar results to ResNet50 but increasing the threshold past 128MB also results in generating smaller messages. For DenseNet201,

we see no impact on message size when changing the threshold. This is also reflected in the throughput measurements being consistent with different thresholds values. Finally, with VGG16 we see mostly message sizes of 16MB and 64MB with a 64MB tensor fusion threshold. It is evident that in this model increasing the threshold directly increases the message sizes used in `MPI_Allreduce`. We suspect that VGG16 has more model parameters than the other models, which results in a larger volume of data being transferred each time the processes determine a global average via `MPI_Allreduce`. This is also reflected in the throughput results for our proposed algorithms, since increasing the thresholds yield a better overall performance for VGG16.

We saw in the micro-benchmark studies that our proposed `MPI_Allreduce` algorithm performs best for large to very large messages, and so when we tune the frameworks to use larger messages, without much affecting their performance, we observe an amplified performance improvement in Horovod with TensorFlow. This shows the significance of our design for Deep Learning workloads.

VII. Conclusion and Future Work

Deep Learning workloads use `MPI_Allreduce` collective extensively, particularly with large messages. In this paper, we addressed the challenges MPI communication libraries have in supporting Deep Learning applications. We proposed a novel intra-socket multi-path point-to-point communication algorithm at the lowest layer of abstraction, UCX, that uses all available NVLink paths concurrently and efficiently strips the messages and utilises multiple GPU streams to enhance the performance. Our proposed approach outperforms UCX by 1.64x. It also improves MPI point-to-point communication performance, which directly uses UCX, by 1.84x.

We then proposed a new hierarchical `MPI_Allreduce` collective design that is NVLink/PCIe-aware and uses in-GPU reduction and multi-path copy for point-to-point communication. We evaluated the performance of our proposed `MPI_Allreduce` collective with Horovod + TensorFlow and various models. We achieved significant performance speedup of up to 2.98x, 3.42x, 3.20x, 1.23x, and 3.21x over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL, respectively.

As for future work, we plan to extend our work in a few ways. We intend to study GPU-based HPC and other Deep Learning workloads that use large MPI point-to-point messages. We plan to devise cluster-wide collective algorithms that could efficiently utilise the underlying algorithms proposed in this paper. We would also like to devise dynamic tuning approaches that would allow this work to be more applicable to a larger set of applications.

Acknowledgement

This research was supported in part by Natural Sciences and Research Council of Canada Grant RGPIN 05389-2016 and Compute Canada. Computations were performed on the Mist supercomputer at the SciNet HPC Consortium. SciNet is funded by the Canada Foundation for Innovation; the Government of Ontario; Ontario Research Fund - Research Excellence; and the University of Toronto. We would like to thank Fei Mao for his technical support on Mist, and Iman Faraji for discussions regarding the detailed implementation of CUDA IPC mechanisms.

References

[1] C. Pearson, A. Dakkak, S. Hashash, C. Li, I.-H. Chung, J. Xiong, and W.-M. Hwu, "Evaluating Characteristics of CUDA Communication Primitives on High-Bandwidth Interconnects," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 209–218. [Online]. Available: https://doi.org/10.1145/3297663.3310299

[2] N. R. Tallent, N. A. Gawande, C. Siegel, A. Vishnu, and A. Hoisie, "Evaluating On-Node GPU Interconnects for Deep Learning Workloads," in *8th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, ser. Lecture Notes in Computer Science, vol. 10724, 2017, pp. 3–21.

[3] A. Li, S. L. Song, J. Chen, X. Liu, N. Tallent, and K. Barker, "Tartan: Evaluating Modern GPU Interconnect via a Multi-GPU Benchmark Suite," in *2018 IEEE International Symposium on Workload Characterization (IISWC)*, 2018, pp. 191–202.

[4] (2021, March) High-Performance Portable MPI. [Online]. Available: http://www.mpich.org

[5] (2021, March) MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE. [Online]. Available: http://mvapich.cse.ohio-state.edu/

[6] (2021, January) Open Source High Performance Computing. [Online]. Available: https://www.open-mpi.org/

[7] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, "UCX: an open source framework for HPC network APIs and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 40–43.

[8] (2021) Message Passing Interface. [Online]. Available: http://www.mpi-forum.org

[9] A. A. Awan, A. Jain, C.-H. Chu, H. Subramoni, and D. K. Panda, "Communication Profiling and Characterization of Deep Learning Workloads on Clusters with High-Performance Interconnects," in *2019 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2019, pp. 49–53.

[10] C.-H. Chu, P. Kousha, A. A. Awan, K. S. Khorassani, H. Subramoni, and D. K. D. K. Panda, "NV-Group: Link-Efficient Reduction for Distributed Deep Learning on Modern Dense GPU Systems," in *Proceedings of the 34th ACM International Conference on Supercomputing*, ser. ICS '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3392717.3392771

[11] R. Rabenseifner, "Optimization of Collective Reduction Operations," in *Computational Science - ICCS 2004*, M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. Dongarra, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 1–9.

[12] P. Sanders, J. Speck, and J. L. Träff, "Two-Tree Algorithms for Full Bandwidth Broadcast, Reduction and Scan," *Parallel Comput.*, vol. 35, no. 12, pp. 581–594, Dec. 2009. [Online]. Available: https://doi.org/10.1016/j.parco.2009.09.001

[13] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *Int. J. High Perform. Comput. Appl.*, vol. 19, no. 1, pp. 49–66, Feb. 2005. [Online]. Available: https://doi.org/10.1177/1094342005051521

[14] J. A. Stuart, P. Balaji, and J. D. Owens, "Extending MPI to Accelerators," in *Proceedings of the 1st Workshop on Architectures and Systems for Big Data*. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2377978.2377981

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

[16] I. Faraji and A. Afsahi, "GPU-Aware Intranode MPI_Allreduce," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 45–50. [Online]. Available: https://doi.org/10.1145/2642769.2642773

[17] ——, "Hyper-Q Aware Intranode MPI Collectives on the GPU," in *Proceedings of the First International Workshop on Extreme Scale Programming Models and Middleware*, ser. ESPM '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 47–50. [Online]. Available: https://doi.org/10.1145/2832241.2832247

[18] (2021) InfiniBand Trade Association. [Online]. Available: http://www.infinibandta.org/

[19] A. R. Mamidala, R. Kumar, D. De, and D. K. Panda, "MPI Collectives on Modern Multicore Clusters: Performance Optimizations and Communication Characteristics," in *2008 Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID)*, 2008, pp. 130–137.

[20] Y. Qian and A. Afsahi, "Process Arrival Pattern Aware Alltoall and Allgather on InfiniBand Clusters," *International Journal of Parallel Programming*, vol. 39, pp. 473–493, 08 2011.

[21] J. L. Träff and A. Rougier, "MPI Collectives and Datatypes for Hierarchical All-to-All Communication," in *Proceedings of the 21st European MPI Users' Group Meeting*, ser. EuroMPI/ASIA '14. New York, NY, USA: Association for Computing Machinery, 2014, pp. 27–32. [Online]. Available: https://doi.org/10.1145/2642769.2642770

[22] Y. Qian, M. Rashti, and A. Afsahi, "Multi-connection and multi-core aware all-gather on InfiniBand clusters," *IASTED International Conference on Parallel and Distributed Computing and Systems*, 01 2008.

[23] C. Chu, K. Hamidouche, A. Venkatesh, A. A. Awan, and D. K. Panda, "CUDA Kernel Based Collective Reduction Operations on Large-scale GPU Clusters," in *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2016, pp. 726–735.

[24] I. Faraji and A. Afsahi, "Design considerations for GPU-aware collective communications in MPI," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 17, p. e4667, 2018, e4667 cpe.4667. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4667

[25] A. A. Awan, J. Bédorf, C. Chu, H. Subramoni, and D. K. Panda, "Scalable Distributed DNN Training using TensorFlow and CUDA-Aware MPI: Characterization, Designs, and Performance Evaluation," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, May 2019, pp. 498–507.

[26] D. Bureddy, H. Wang, A. Venkatesh, S. Potluri, and D. K. Panda, "OMB-GPU: A Micro-Benchmark Suite for Evaluating MPI Libraries on GPU Clusters," in *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface*, ser. EuroMPI'12. Berlin, Heidelberg: Springer-Verlag, 2012, p. 110–120. [Online]. Available: https://doi.org/10.1007/978-3-642-33518-1_16

[27] Q. Anthony, A. A. Awan, A. Jain, H. Subramoni, and D. K. D. Panda, "Efficient Training of Semantic Image Segmentation on Summit using Horovod and MVAPICH2-GDR," in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020, pp. 1015–1023.