Accelerating Deep Learning using Interconnect-Aware UCX Communication for MPI Collectives

Yıltan Hassan Temuçin, AmirHossein Sojoodi, Pedram Alizadeh, Benjamin Kitor, and Ahmad Afsahi
Queen's University, Canada

***Abstract*—Deep Learning workloads on modern multi-GPU nodes are highly dependent on intra-node interconnects, such as NVLink and PCIe, for high-performance communication. In this paper, we take on the challenge to design an interconnect-aware multi-path GPU-to-GPU communication using UCX to utilise all available bandwidth for both NVLink-based systems and those that use a mixture of NVLink and PCIe. Our proposed multi-path data transfer mechanism pipelines and stripes the message across multiple intra-socket communication channels and memory regions to achieve 1.84x higher bandwidth for Open MPI on NVLink-based systems and 1.23x on NVLink and PCIe systems. We then utilise this mechanism to propose a three-stage hierarchical, pipelined `MPI_Allreduce` design as well as a flat pipelined two-stage algorithm for two different node topologies. For large messages, our proposed algorithms achieve a high speedup when compared to other MPI implementations. We also observe significant speedup for the proposed `MPI_Allreduce` with Horovod + TensorFlow with a variety of Deep Learning models.**

■ **HIGH PERFORMANCE COMMUNICATION FOR DEEP LEARNING** has become increasingly important in recent years due to the large growth in distributed Deep Learning applications. These workloads put tremendous pressure on the communication subsystem of computing systems and their performance is significantly affected by the underlying interconnect and communication runtime. The usage of GPU accelerators in High-Performance Computing (HPC) systems are becoming ever more prevalent with multi-GPU computing nodes. These nodes are equipped with sophisticated intra-node interconnects such as Nvidia's high-speed NVLink and PCIe to provide interconnectivity for GPU communication. Majority of HPC systems use the Message Passing Interface (MPI) [1] for parallel programming. MPI supports point-to-point, partitioned point-to-point, collective, and remote memory access (RMA) communication operations. MPI collectives, in particular, `MPI_Allreduce` and `MPI_Bcast` play a crucial role in the performance of MPI applications, including Deep Learning workloads. Existing designs use a single NVLink or PCIe communication path to transfer data. In this paper, we take on the challenge to design efficient point-to-point GPU communication using all available communication paths. We then utilise our point-to-point multi-path scheme to design two different intra-node `MPI_Allreduce` collectives 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 multiple available communication channels. Our approach achieves 1.84x and 1.23x higher bandwidth for Open MPI + UCX for Mist and Cedar, respectively.
- On Mist, we propose a 3-stage hierarchical, pipelined `MPI_Allreduce` collective that utilises the new multi-path copy mechanism for intra-socket data transfers, while dynamically selecting between NVLink and PCIe channels. Our results show a speedup of 1.38x to 15.63x over other MPI implementations.
- On Cedar, we propose a 2-stage flat, pipelined `MPI_Allreduce` collective design that utilises the new multi-path copy mechanism for data transfers. Our experimental results show a speedup of up to 104.1x, 110.9x, and 1.06x against Open MPI + UCX, MVAPICH2, and NCCL, respectively.
- We evaluate our proposed collective with Horovod + TensorFlow. For VGG16 and ResNet50, we observe up to 3.42x and 1.57x speedup on Mist, respectively. On Cedar, we achieve 5.74x speedup for DenseNet201.

Open MPI + UCX

UCX (Unified Communication X) is an RDMA-based point-to-point communication library for modern low latency, high bandwidth interconnects [2]. The Message Passing Interface (MPI) is the most popular programming model used in high-performance computing. When using point-to-point communication in Open MPI, we are directly using UCX. Collective communication is implemented via send/recvs of UCX. Open MPI supports various flat algorithms for `MPI_Allreduce`. GPU support follows a general approach that involves staging the GPU data into the host buffer and leveraging the CPU-based MPI routines.

Challenges in MPI-based DL

Research has shown the performance of GPU based Deep Learning workloads, such as Horovod + TensorFlow, are highly dependent on the performance of collective communication operations that use very large messages [3]. Figure 1 presents the distribution of calls using CPU and GPU buffers for various Deep Learning models. We observe that a total 17-83% of training time is spent in `MPI_Allreduce`, and up to 80% of runtime when using GPU buffers. As collectives are designed on top of point-to-point communication, there could be opportunities to improve the `MPI_Allreduce` performance by directly targeting the intra-node GPU-based point-to-point communication for large messages.

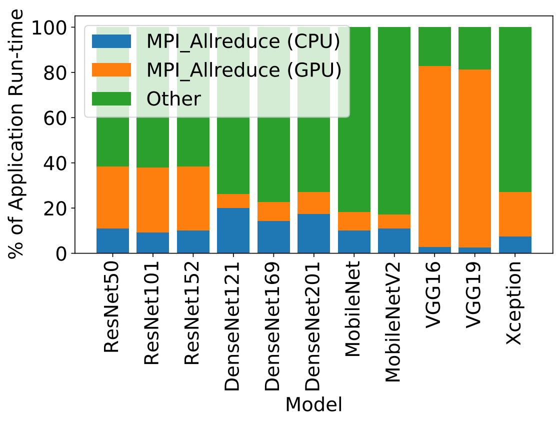


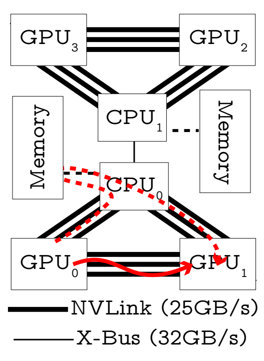
Figure 1: Percentage of runtime spent in `MPI_Allreduce` when called with CPU or GPU buffers for various Deep Learning models.

Looking over Figure 2 one can realise that each GPU can be reached through two distinct paths on contemporary multi-GPU nodes. For Mist, we only have NVLink communication channels. Whereas on Cedar, we have one NVLink and one PCIe path available for data transfer.

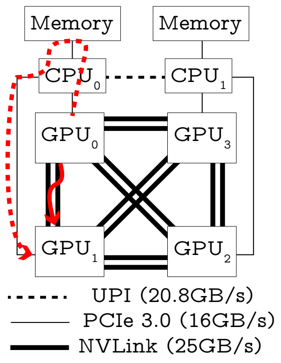
Current MPI implementations send data directly from GPU_0 to GPU_1 via the NVLink connecting the two devices. This results in the NVLinks or PCIe connected to the host to be idle during data transfers. This motivated us to utilise all available intra-socket paths to increase point-to-point bandwidth between GPU pairs.

The Multi-Path Communication

Our multi-path copy design [4] partitions send buffer into two pieces and transfers each data segment via an independent data channels, as shown in Figure 2. We first calculate the size of the



(a) Mist



(b) Cedar

Figure 2: Hardware Topology of HPC Systems

partition we are sending via the host, and copy it into host memory via NVLink or PCIe, depending on the platform. Then, we copy from the host memory to the destination GPU. In parallel, we copy the data from the source GPU directly to the destination GPU. For transferring the data via the host, we stage our transfers in page-locked host memory accessible by the GPUs. We further split the host data transfer into multiple chunks that are placed on individual streams to allow for pipelining between D2H and H2D copies. For our study, we experimented with different chunk counts and found the optimal bandwidth for each message size. As we use a mixture of host-staged and D2D copies, the data volume which we send via each path must be tuned to maximise aggregate bandwidth. We roughly send around 25-30% of the message via the host and the remainder directly to the adjacent GPU on Mist and slightly less on Cedar. We record the optimal number of chunks and message distribution for each message size in a static tuning table. For smaller message sizes we use a single stream and for larger messages we use up to 8 streams.

The Hierarchical Intra-Node GPU-based MPI_Allreduce

We propose a three-stage intra-node hierarchical GPU-based `MPI_Allreduce` to use our multi-path design with the goal of impacting Deep Learning workloads on multi-GPU nodes:

1) Intra-socket Reduce Stage: All intra-socket GPUs send their data to their intra-socket GPU leaders (GPU_1 and GPU_2, without loss of generality) to reduce their data.
2) Inter-socket Reduce stage: The intra-socket GPU leaders exchange and reduce their data.
3) Intra-socket Broadcast stage: The intra-socket GPU leaders send the reduced data back to their intra-socket GPUs.

Open MPI uses a CPU based allreduce even when the data is resident in the GPU global memory. We modified the existing Open MPI implementation of `MPI_Allreduce` to remove host-staging of data and directly transfer GPU-resident data using UCX's CUDA IPC features. As our new design no longer copies data to the host, we use a simple CUDA kernel for reduction. To further optimise this collective, we overlapped inter- and intra-socket communication, and dynamically switched between PCIe and NVLink. Full details about this algorithm on Mist can be found in [4]. For Cedar, the NVLink topology is flat, therefore the inter-socket NVLinks remain idle when using this algorithm.

The Flat Intra-Node GPU-based MPI_Allreduce

The proposed hierarchical intra-node collective is non-optimal on Cedar due to its fully-connected GPU topology. We develop a new 2-stage algorithm to fully utilise the communication channels, by performing a segmented reduction on all NVLink channels, followed by a broadcast operation. We use the same GPU-based kernel reduction as before.

1) Reduce Stage: Each GPU sends a distinct quarter of its data to every other GPU to be reduced.
2) Broadcast Stage: Each GPU will then broadcast its reduced data to all other GPUs.

If this algorithm is implemented naively, no communication would occur during the GPU kernel reduction. To increase network usage during reduction, we further pipelined this algorithm, by subdividing the buffers into more than 4 chunks described above. Once a GPU receives the data, it posts a send for the next chunk of data, then reduces the current data chunk. We then applied our proposed multi-path copy for intra-socket communication. To avoid interference among multiple transactions, we did not apply our multi-path copy to inter-socket communication.

The Hierarchical Multi-Node GPU-based MPI_Allreduce

The previous intra-node collective designs are only applicable to a single node. To illustrate the applicability of our multi-path copy to multi-node clusters, we modify the proposed hierarchical intra-node GPU-based `MPI_Allreduce` algorithm so that it can scale. First, we create an intra-node and an inter-node communicator when the collective is called. We cache these communicators so that we do not repeatedly create them at run-time. Step 1 of the intra-node algorithm is executed as before using the intra-node communicator. Step 2 is modified so that the intra-socket leaders execute a reduce-scatter operation, then call pre-existing Open MPI `MPI_Allreduce` collectives using the inter-node communicators. Next, an allgather is executed between the intra-socket leaders. Finally, Step 3 is executed as before. This approach could also be used for our flat intra-node `MPI_Allreduce` on Cedar with minor differences. Although this design scales our multi-path copy to multiple nodes, it is fairly naive as we have not considered inter-node network features or software techniques such as intra/inter-node pipelining in our design.

Performance Evaluation and Analysis

Experiments were conducted on two systems in Compute Canada, Cedar at WestGrid and Mist at the SciNet HPC Consortium. **Cedar** is a Dell C4140 server with dual-socket 16-core Intel Silver 4216 Cascade Lake with four 32GB V100-SXM2 GPUs and 192GB of memory. All GPUs are connected to one another with two NVLinks. **Mist** is an IBM POWER9 AC922 machine with two sockets for a total of 32 cores and 382GB of memory, and four Nvidia V100-SXM2 (32GB) GPUs, with three NVLinks between intra-socket GPUs and to the host processors. The NVLink/PCIe topologies of these systems are shown in Figure 2. On both systems, we have used Open MPI 4.0.4rc2, UCX 1.8.0, and NCCL 2.5.6. For our studies on Mist, we have also used Open MPI + HPC-X (with UCX and HCOLL) from HPC-X v2.7, Spectrum-MPI 10.3.1, MVAPICH2-GDR 2.3.5, 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. On Cedar, we used MVAPICH2 v2.3.6, and Horovod 0.18.3 with TensorFlow 1.13.1. We do not have ResNet152 results on Cedar as the model was not implemented for this TensorFlow version.

UCX Put results using the multi-path copy for intra-socket data transfers on Mist and Cedar can be seen in Figure 3. On Mist, we can see the peak bandwidth of 120GB/s (1.67x) when using the proposed multi-path copy mechanism. On Cedar, we only see a speedup of 1.18x as our bandwidth increases from around 45GB/s to 54GB/s. The difference in the initial bandwidth measurements stem from Cedar having 2 NVLinks between two GPUs whereas Mist has 3 NVLinks, providing roughly 25GB/s extra bandwidth. The other difference we can see in Figure 2 is that for our host-staged copies we use PCIe on Cedar and NVLink on Mist, with a theoretical limit of 16GB/s and 75GB/s, respectively. Therefore, we should expect smaller performance gain on Cedar. On Mist we obtain 80% of our theoretically expected bandwidth and on Cedar we obtain 82%. So, our multi-path copy design is consistent across the two platforms, but the limitations are stemming from hardware.

MPI point-to-point bandwidth was evaluated using the Ohio State University Micro-Benchmark (OMB) suite. Figure 4 shows the uni-directional bandwidth results with a window size of 64 between two GPUs on the same socket for both Mist and Cedar. On Mist, we achieve a peak bandwidth of 134GB/s (1.84x), which is slightly higher than the UCX test due to the larger window size that can better saturate the NVLinks. On Cedar, we obtain a peak bandwidth of 60GB/s (1.23x) using our multi-path copy. As Open MPI is implemented on top of UCX, we observe the same hardware limitations that we saw in our UCX results, as the performance improvement on Cedar is much less than Mist.

Intra-Node MPI_Allreduce tests were configured with data 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 5 and Fig-

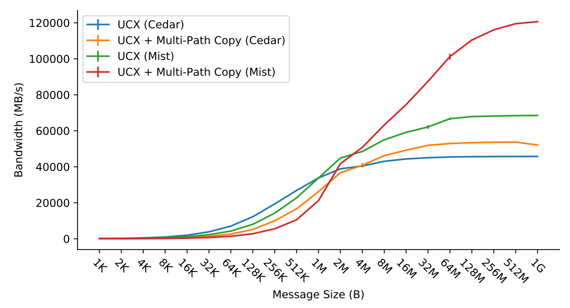


Figure 3: UCX Put bandwidth comparing P2P to the multi-path copy on different platforms

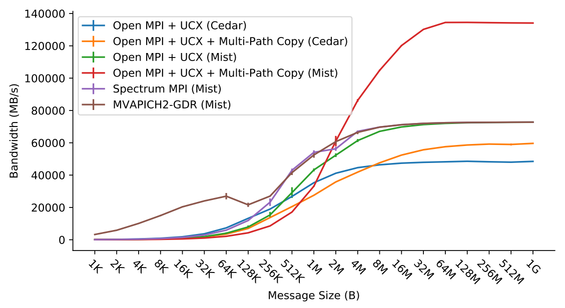


Figure 4: OMB results for MPI unidirectional bandwidth on different platforms

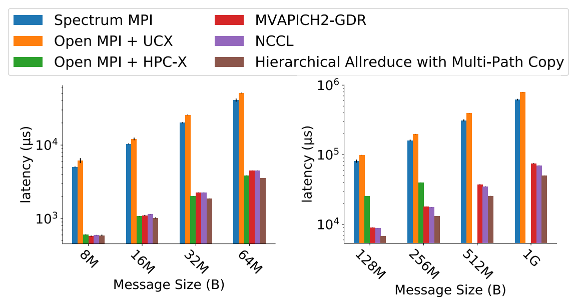
ure 6 presents the micro-benchmark results for `MPI_Allreduce` for message sizes in 8MB-1GB range on Mist and Cedar, respectively. For all tests, we have used four processes on a single node with each bound to a single GPU.

On Mist, our design shows significant performance improvements over Open MPI + UCX and Spectrum MPI. We believe this is because `MPI_Allreduce` in these libraries use a CPU based reduction even for GPU resident data. Our results also outperform Open MPI + HPC-X for all message sizes and NCCL and MVAPICH2-GDR for message sizes greater than 8MB.

The proposed hierarchical intra-node 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. For 1GB messages, we outperform Spectrum MPI, Open MPI + UCX, MVAPICH2-GDR, and NCCL 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.

On Cedar, our proposed algorithms outperform both Open MPI + UCX and MVAPICH for all message sizes. We can see the benefit of our Flat Allreduce compared to Open MPI + UCX on Cedar; whereas the Hierarchical Allreduce would not efficiently use the hardware resources for large GPU messages. At 64MB, our Flat Allreduce gains a speedup of 104.1x, 110.9x, 2.77x, and 1.06x compared to Open MPI + UCX, MVAPICH2, Hierarchical Allreduce, and NCCL, respectively. Our improvements over NCCL are small on Cedar, as NCCL is already well optimised for V100-SMX2 package. We see that using a CPU based reduction for GPU buffers has a much larger impact on the performance of `MPI_Allreduce` on Cedar than on Mist. This is because Mist has NVLink connections to the host, whereas Cedar has only PCIe connections to the host. Thus, host stage copies are a larger bottleneck on PCIe platforms.

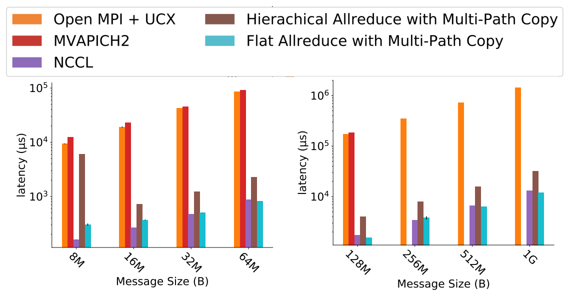
Multi-Node MPI_Allreduce hierarchical algorithm was scaled up to four nodes on Mist as



(a) Large

(b) Very Large

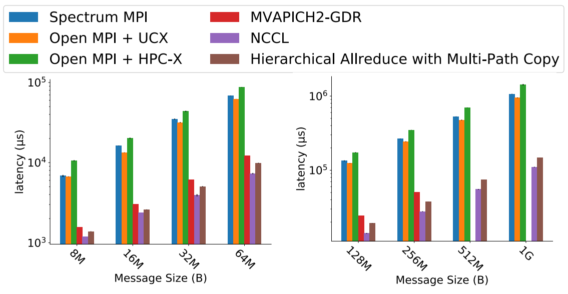
Figure 5: Intra-Node `MPI_Allreduce` results on Mist



(a) Large

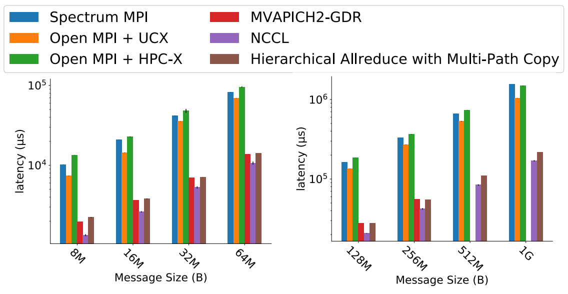
(b) Very Large

Figure 6: Intra-Node `MPI_Allreduce` results on Cedar



(a) Large (b) Very Large

Figure 7: Multi-Node `MPI_Allreduce` on Mist with two Nodes



(a) Large (b) Very Large

Figure 8: Multi-Node `MPI_Allreduce` on Mist with four Nodes

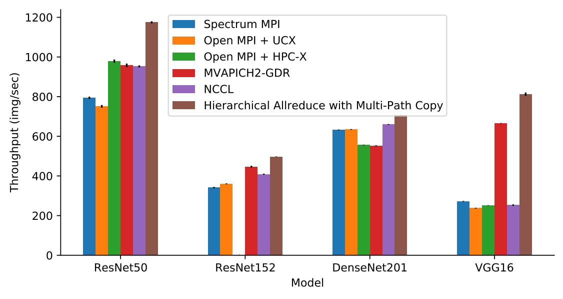
that was the maximum available nodes for users on this cluster. As shown in Figure 7 and Figure 8, we still outperform both Spectrum MPI and Open MPI + UCX. This is due to the same reasons as the intra-node collective but also because Open MPI + UCX uses a flat algorithm across multiple nodes. For two nodes, we outperform MVAPICH2-GDR for all messages sizes, and at 64MB we see a 1.18x speedup. Our proposal is comparable to MVAPICH2-GDR at four nodes. The results we obtained for HPC-X for multiple nodes is less performant than what we would expect given the intra-node results. We experimented with many different enviroment variables to tune HPC-X but this was the best performance we were able to get on this micro-benchmark. All MPI implementations fall short to NCCL, as it takes advantage of the inter-node network in its collective design. Our proposed design did not also consider this.

Intra-Node Horovod with TensorFlow results with four Deep Learning models, ResNet50, ResNet152, DenseNet201, and VGG16 using our proposed design can be seen in Figure 9a for Mist. We varied the Horovod tuning parameter `HOROVOD_FUSION_THRESHOLD` and found the best throughput at 1GB fusion buffer with a batch size of 32. For ResNet50, we see a throughput speedup of up to 1.48x, 1.56x, 1.20x, 1.23x, and 1.23x over Spectrum MPI, Open MPI + UCX, Open MPI + HPC-X, MVAPICH2-GDR, and NCCL, respectively. A modest performance speedup of 1.12x, 1.11x, 1.27x, 1.28x, and 1.07x is seen over the above libraries for DenseNet201, respectively. VGG16 shows the highest performance speedup of 2.98x, 3.42x, 3.20x, 1.22x, and 3.21x over the above libraries, respectively. With ResNet152, we observe a throughput speedup of 1.46x, 1.38x, 1.11x, and 1.21x over Spectrum MPI, Open MPI + UCX, MVAPICH2-GDR, and NCCL, respectively. We were unable to obtain results for Open MPI + HPC-X as this model would cause the application to segfault.

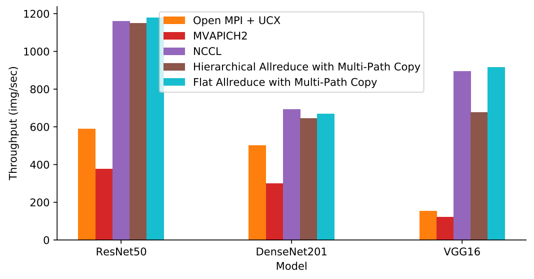
On Cedar, as shown in Figure 9b, our Flat Allreduce achieves 2.00x, 3.14x, 1.02x, and 1.03x speedup for ResNet50 over Open MPI + UCX, MVAPICH2, NCCL, and Hierarchical Allreduce, respectively. For DenseNet201, we see an increase in throughput of 1.33x, 2.23x, 0.97x, and 1.04x over the above-mentioned implementations, respectively. For VGG16, we achieve 5.96x, 7.51x, 1.02x, and 1.35x speedup over these libraries, respectively. As in our point-to-point and collective micro-benchmark studies on Cedar, a smaller performance improvement is also observed at the application layer.

Overall, it is clear from these results that the performance improvement for Horovod + TensorFlow with the proposed `MPI_Allreduce` algorithms is significant, but fairly dependent on the model and platform. The results are fairly consistent with our expectations from Figure 1.

To investigate this further, Figure 10 presents the frequency of message sizes used by `MPI_Allreduce` at 1GB fusion buffer during Horovod's execution. We see that GPU message size for `MPI_Allreduce` is model dependent. VGG16 generates very large messages than the other models which could indicate why our proposed collective has the largest impact on this model on Mist. The same could be said for DenseNet201, where this model uses a lot of



(a) Mist



(b) Cedar

Figure 9: Intra-Node Horovod + TensorFlow throughput with different models

medium sizes messages that our collective does not target.

We saw in the micro-benchmark studies that our proposed `MPI_Allreduce` algorithms perform 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.

Multi-Node Horovod with TensorFlow show large improvements results for multiple nodes in Figure 11 for ResNet50, ResNet152 and VGG16, which are consistent with the single node results. We still see some improvement for DenseNet201, but it is fairly modest. We had issues running Horovod and Spectrum MPI with multiple nodes but the expected results would be similar to those of Open MPI. As we saw in the multi-node micro-benchmark results, NCCL outperforms all MPI implementations at the application layer as well.

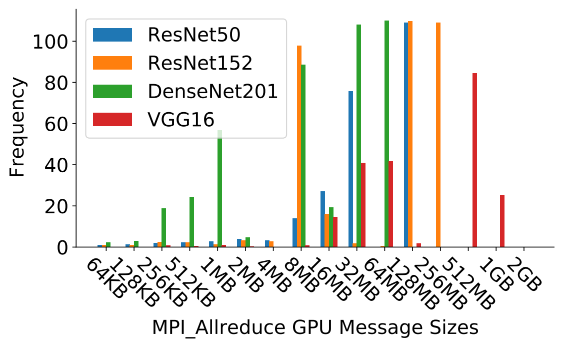
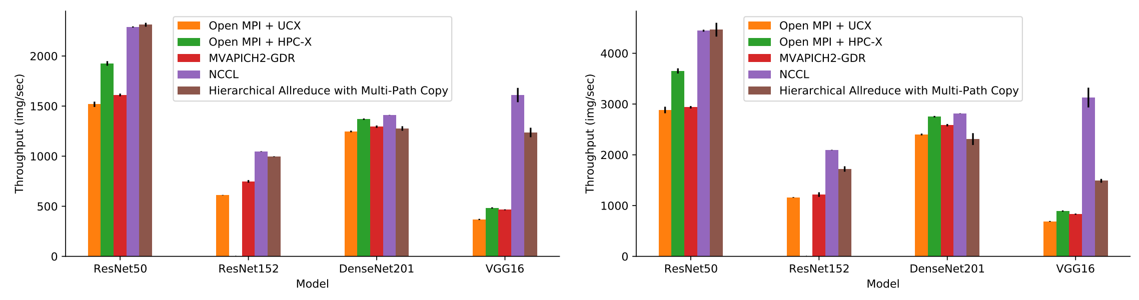


Figure 10: `MPI_Allreduce` GPU message sizes in Horovod + TensorFlow with 1GB fusion buffer

CosmoFlow Application results were collected on Mist to see if our proposal could apply to a wider range of applications. CosmoFlow is an application which trains a DNN on N-body cosmology simulation data to model the universe. CosmoFlow is a more realistic Deep Learning workload compared to the throughput benchmarks. We ran the application for 100 epochs, and the results are shown in Figure 12. For the mean epoch time, we see a small difference between the various MPI implementations as all results are within 7.2% of one another. Although we had a large difference in `MPI_Allreduce` latency in Figure 5, these differences were not reflected by the application. We investigated further by measuring the communication time spent in `MPI_Allreduce` using GPU buffers, and these results are shown in Figure 12a. Results for NCCL are not shown, as they could not be measured using our PMPI profiler. We found that around 10% of application run-time was spent in `MPI_Allreduce` with GPU. This helps explain why we see both our Hierarchical Allreduce and MVAPICH2-GDR spend less time in communication than the default Open MPI, but this does not translate to any end-to-end performance improvements. It should also be mentioned that unlike the Horovod benchmarks, modifying the fusion buffer had minimal impact on runtime.

Related Work

There is an abundance of research regarding collective communications, but in recent years workloads have begun to rely much more on GPU accelerators. NVIDIA introduced Inter-Process Communication (IPC) with CUDA 4.1, and Faraji and Afsahi investigated CUDA IPC within the



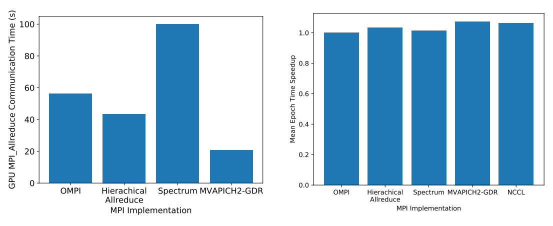
(a) Two Nodes (b) Four Nodes

Figure 11: Multi-Node Horovod + TensorFlow throughput with different models on Mist

context of an MPI_Allreduce collective design [5]. In [6], the same authors 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. For a set of communications within a collective for each process, they used a combination of the host-staged and CUDA IPC copies for inter-process communications by efficiently leveraging the MPS and Hyper-Q feature and provided enhancement for medium and large messages. In [7], 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 [6] and [7] 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. 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.

With the emergence of modern high-bandwidth interconnects such as NVLink, research has focused on their impact on communication performance. Tallent et al. presented the impact of NVLink and PCIe interconnects on Deep Learning workloads [8]. In [9], Li et al. evaluated such interconnects with a multi-GPU benchmark suite.

For applications in Deep Learning, Chu et al. [10] showed that GPU-based reduction was more performant than host-based reduction for large messages in large clusters. In [11], Awan et al. focused on MPI_Allreduce and moved the computational part of the collective to GPUs, resulting in significant improvements in Horovod. Alongside kernel-based reduction, Chu et al. studied the underutilisation of NVLinks [7] by taking the physical topology of the system into account. Our proposals differs from these works as we target the underutilised NVLinks at the point-to-point layer and our collective design leverages multiple communication during its execution.



(a) Communication Time (b) Mean Epoch Time

Figure 12: CosmoFlow Results for various MPI implementations and NCCL on Mist

Conclusion and Future Work

In this paper, we addressed the challenges MPI communication libraries have in supporting Deep Learning applications that use MPI_Allreduce collective with large messages extensively, 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 or PCIe paths concurrently and efficiently strips the messages and utilises multiple GPU streams to enhance the performance. We evaluated this design on two different platforms and observed up to 1.84x and 1.23x improvement in point-to-point bandwidth, respectively. We then proposed two new `MPI_Allreduce` collective designs that are interconnect-aware and use in-GPU reduction and multi-path copy for point-to-point communication. We evaluated the performance of our proposed `MPI_Allreduce` collective on Horovod + TensorFlow with various models on two different topologies and achieved up to 5.75x higher throughput.

For future work, we would like to extend our studies to the new Nvidia Ampere A100 GPUs. We also plan to further develop our cluster-wide collective algorithm to consider pipelining and utilize advanced inter-node networking features. 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. This research was enabled in part by support provided by WestGrid (www.westgrid.ca), SciNet (www.scinethpc.ca) and Compute Canada Calcul Canada (www.computecanada.ca). We would like to thank Fei Mao and Bruno C. Mundim for their technical support on Mist, and Iman Faraji for discussions regarding the detailed implementation of CUDA IPC mechanisms.

REFERENCES

1. (2021) Message Passing Interface. [Online]. Available: http://www.mpi-forum.org
2. 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.
3. 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.
4. Y. H. Temucin, A. Sojoodi, P. Alizadeh, and A. Afsahi, “Efficient Multi-Path NVLink/PCIe-Aware UCX based Collective Communication for Deep Learning,” in *2021 IEEE Symposium on High-Performance Interconnects (HOTI)*, 2021.
5. 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
6. ——, “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
7. 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
8. 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.
9. 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.
10. 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.
11. 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.

Yıltan Hassan Temuçin is currently pursuing a PhD degree at Queen's University, Canada. He also received his B.A.Sc. and M.A.Sc. degrees from

Queen's University. His research interests are in high performance communication for deep learning applications, in particular, GPU-Aware and Stream-Aware MPI implementations. Contact him at yiltan.temucin@queensu.ca

AmirHossein Sojoodi is currently pursuing his Ph.D. degree in Computer Engineering at Queen's University in Canada. He received his B.Sc. and M.Sc. degrees in Software Engineering from Shiraz University, Iran. His research interests are high-performance communications, GPU computing, and distributed processing. Contact him at amir.sojoodi@queensu.ca

Pedram Alizadeh received his M.A.Sc. degree in Electrical and Computer Engineering at Queen's University, Kingston, ON, Canada. He received his B.Sc. degree in Electrical Engineering at K. N. Toosi University of Technology, Tehran, Iran. His research interests are in high-performance computing such as scalable and efficient communication for Deep Learning applications, process arrival pattern aware collective communications, and high-speed interconnects. Contact him at 18pm7@queensu.ca.

Benjamin Kitor is currently pursuing his M.A.Sc. degree in Electrical and Computer Engineering at Queen's University, Kingston, ON, Canada. He received his B.A.Sc. degree in Computer Engineering from Queen's University. His research interests are high-performance communication, and topology-aware communication for Deep Learning. Contact him at ben.kitor@queensu.ca

Ahmad Afsahi Ahmad Afsahi is currently a Full Professor in the Department of Electrical and Computer Engineering at Queen's University, Kingston, ON, Canada. He received his Ph.D. degree in Electrical Engineering from the University of Victoria, BC in 2000, and his M.Sc. and B.Sc. degrees in Computer Engineering from Sharif University of Technology and Shiraz University, Iran, respectively. His main research activities are in the areas of accelerated cluster computing, communication runtime and system software, high-speed interconnects and communication subsystems, high-performance communication for Deep Learning, high-performance computing, parallel programming models, and workload characterization. His research has earned him a Canada Foundation for Innovation Award and an Ontario Innovation Trust. He has published over 75 research papers, winning several best paper awards. Dr. Afsahi is a Senior Member of IEEE, a Member of ACM, and a licensed Professional Engineer in the province of Ontario. Contact him at ahmad.afsahi@queensu.ca