A Performance Analysis of GPU-Aware MPI Implementations Over the Slingshot-11 Interconnect

Michael Beebe*, Rahulkumar Gayatri†, Kevin Gott†,
Adam Lavely†, Muhammad Haseeb‡, Brandon Cook†, and Yong Chen*

*Texas Tech University, Lubbock, Texas, USA
†Lawrence Berkeley National Laboratory, Berkeley, California, USA
‡NVIDIA Corporation, Santa Clara, California, USA

***Abstract*—This study evaluates the performance of three GPU-aware MPI implementations—Cray MPICH, MPICH, and Open MPI—on the Slingshot-11 interconnect using the Perlmutter supercomputer. We use the OSU microbenchmarks to evaluate the bandwidth and latency of specific MPI routines, and two Exascale Computing Projects - LAMMPS and WarpX to evaluate application performance. Results confirm that Cray MPICH outperforms the others over Slingshot-11, significantly influencing HPC application efficiency by leveraging proprietary hardware mechanisms. This research assists domain scientists in selecting MPI libraries, enhancing application performance on Slingshot-11 systems and contributing to future software and hardware optimization studies.**

***Index Terms*—GPU-Aware MPI, Slingshot-11, Perlmutter**

I. Introduction

Message Passing Interface (MPI) has been ubiquitous in HPC for decades [1]. However, the performance portability of MPI implementations across architectures and network technologies can vary significantly, impacting the execution times of communication-bound applications. This necessitates MPI library vendors to continually optimize their implementations for newly introduced interconnects and accelerator devices. The impending demise of Moore's law has resulted in a paradigm shift toward heterogeneous computing, leading to an increased use of accelerators and tightly coupled hardware/software co-design. MPI implementations are adapting this trend by providing GPU-aware support. With GPU-awareness enabled, MPI implementations can recognize when a buffer (pointer) resides on GPU memory and can engage mechanisms such as NVIDIA GPUDirect [2] to initiate direct GPU-to-GPU communications. This allows the GPU-aware MPI implementations to avoid the extra steps in traditional MPI data flows that involve transferring data from GPU to CPU at source, communicating it between CPUs, and transferring it to the GPU at destination. With rapid evolution in interconnect technology and the number of MPI settings available to improve performance for different interconnects, it can be challenging for scientists to choose the most performant MPI implementation for their application. In this study, we evaluate the performance of point-to-point and collective communications between Cray MPICH, MPICH, and Open MPI over Slingshot-11 using the OSU microbenchmarks [3], and evaluate application performance differences with two Exascale Computing Project applications - LAMMPS [4] and WarpX [5]. Based on our results, we demonstrate that the choice of MPI implementation can have a significant impact on the communication performance of HPC applications, showing that programmers and scientists can benefit from resources that help make informed decisions in MPI selection when deploying their applications across supercomputing facilities.

II. Background

A. Perlmutter

The system used to evaluate our selected MPI implementations is the Perlmutter supercomputer housed at the National Energy Research Scientific Computing Center (NERSC). As shown by the node specifications depicted in Fig. 1, each node is equipped with four HPE Slingshot-11 Cassini NICs per node, one 64-core hyper-threading-enabled AMD EPYC 7763 processor, 256GB of DDR4 DRAM spread across eight slots, and four 40GB HBM2e NVIDIA A100 GPUs. Each NIC and GPU is connected to the CPU by way of a fourth generation PCIe bus, and there are twelve third generation NVLink links between each pair of GPUs boasting 25GB/s/direction for each link. The network is configured with a 3-hop dragonfly topology.

B. Slingshot-11

HPE's Slingshot interconnect [6] is designed for HPE Cray supercomputers and HPC clusters, targeting exascale computing needs in simulation, modeling, AI, and data analytics. A key feature is the high-radix 64-port switch, which reduces network diameter and lowers latency compared to traditional lower-radix designs [7]. Slingshot-11 includes an enhanced congestion control mechanism with notable quality-of-service (QoS) capabilities, ensuring predictable performance by dynamically managing and prioritizing data traffic. Its adaptive routing continuously optimizes data pathways based on real-time network conditions, ensuring efficient data transfer even under high network traffic.

C. MPI Libraries

Cray MPICH [8] is optimized for Cray supercomputers and the Slingshot interconnect, supporting unique network topologies like Dragonfly. This tight hardware-software integration minimizes communication latencies, offering performance benefits over general-purpose MPI implementations.

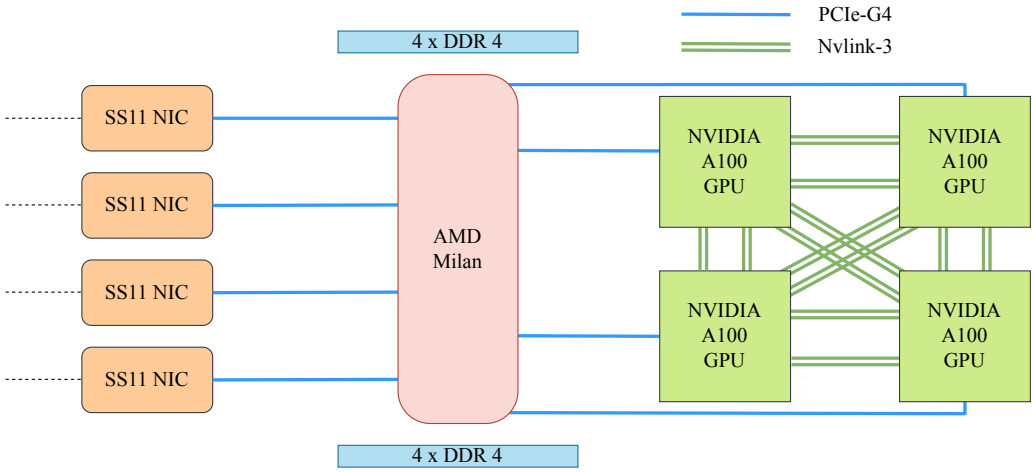

Fig. 1: Perlmutter GPU Node Specification

MPICH [9], maintained by Argonne National Laboratory, is a standard-compliant, scalable MPI implementation. Its extensible framework integrates various device drivers, optimizing performance across diverse platforms and interconnects, making it a popular choice in supercomputing environments. Open MPI [10] is a flexible, open-source MPI implementation with a modular architecture. It supports various network plugins (BTLs) and offers the MCA (Modular Component Architecture) framework, enabling tailored communication pathways and portability across different systems, from high-speed interconnects to common Ethernet.

D. OSU Microbenchmarks

To obtain performance metrics for our comparisons between our MPI implementations, we use the OSU microbenchmarks [3] version 7.1, a widely used benchmark suite that is used to measure point-to-point, multi-pair, and collective communication latency. We examine point-to-point latency and bandwidth in III-A, and blocking collective operations in III-B. We investigate message sizes ranging from 4 bytes to 16MB to cover a wide variety of uses cases and application behaviors. From these results, we can predict which MPI implementation will perform the best for communication-bound applications based on prevalence of each MPI routine as a constraint within that application. In the interest of evaluating each of our selected MPI libraries' GPU-aware implementation, we configured the OSU microbenchmarks such that messages are being transmitted to and from GPU buffers in an inter-node fashion. The results reported in this study are the averages of five runs.

E. Scientific Applications

1) LAMMPS

The Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [4] is a highly versatile and widely used open-source molecular dynamics simulation package maintained by Sandia National Laboratory. LAMMPS facilitates the modeling of atomic, molecular, and polymetric systems across a diverse range of conditions. Much work has been done in LAMMPS to optimize communications.

2) WarpX

WarpX [5] is an electromagnetic Particle-In-Cell (PIC) code that is designed to model and simulate advanced accelerator concepts and plasma-based particle accelerators. The code is built on the AMReX [11] adaptive mesh refinement (AMR) library, allowing it to efficiently handle complex geometries and dynamically refine regions of interest.

III. Evaluation & Analysis

In this section, we discuss the benchmark and application performance of each tested GPU-aware MPI implementation. Table I shows the software versions used when conducting the tests.

TABLE I: Software Versions

Software	Version
GNU Compiler	11.2.0
CUDA	11.7
Libfabric	1.15.2.0
Cray MPICH	8.1.25
MPICH	4.1.1
Open MPI	5.0
OSU Microbenchmarks	7.1.1
LAMMPS	2022.11.03
WarpX	23.04

A. Point-To-Point MPI Operations

For our first set of evaluations, we examine point-to-point latency and bandwidth for our three selected MPI implementations whereby messages are sent from a GPU buffer on one node and received by a GPU buffer on another node. The latency tests are carried out in a ping-pong fashion. Many iterations of this ping-pong test are carried out and the average one-way latency numbers are reported. The bandwidth tests

are carried out by sending a fixed number of back-to-back messages from GPU buffers and then waiting for a reply from the GPU buffers on the receiver. The receiver sends the replies back only after receiving all the sent messages. This process is repeated for one thousand iterations and the bandwidth is calculated using the elapsed time. The objective of the bandwidth test is to determine the maximum sustained data rate that can be achieved on the network level. Thus, non-blocking sends and receives are used for the point-to-point bandwidth test. All point-to-point tests utilize two nodes, and one process per node.

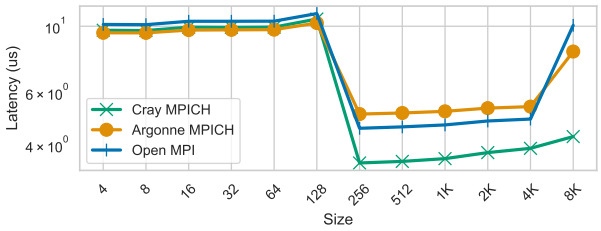

Fig. 2: Point-to-Point Latency (Small Message Sizes)

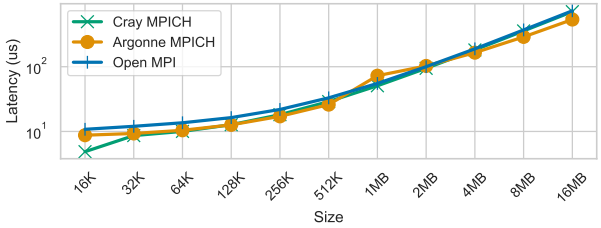

Fig. 3: Point-to-Point Latency (Large Message Sizes)

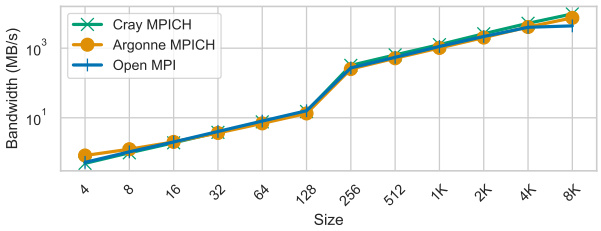

Fig. 4: Point-to-Point Bandwidth (Small Message Sizes)

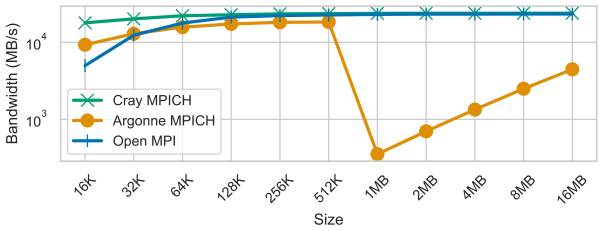

Fig. 5: Point-to-Point Bandwidth (Large Message Sizes)

We find that the latencies for all message sizes are similar, and when there are differences, the Cray MPICH tends to provide the best performance. As shown in Figures 2 and 3, latencies for all three implementations fall when increasing from 128 byte to 256 byte message sizes, with Cray MPICH having the most pronounced drop. As we increase the message size from 256 bytes, latencies for all three implementations steadily rise with Argonne MPICH and Open MPI exhibiting a large spike moving from 4K messages to 8K messages not seen in the Cray MPICH data. For larger message sizes, all three implementations exhibit similar point-to-point latency. Similarly for point-to-point bandwidth, the three MPI implementations are fairly similar outside of a few differences. All three implementations have a near order of magnitude jump when going from 128 to 256 bytes, and reach similar overall peak bandwidth values. The Argonne MPICH shows a drop in bandwidth when moving from a 512K message size to a 1MB message, and a steady rise and bandwidth thereafter shown in Figure 5.

B. Collective MPI Operations

For the next set of evaluations, we examine the latencies of several blocking collective MPI routines. These tests are carried out by performing collective operations with four processes per node (one for each GPU) and measure the average latency of each operation over one thousand iterations.

1) Strong Scaling

First, we perform a set of tests where we use a fixed message size of 16MB and an increasing number of nodes. The tests are for `MPI_Gather` in Figure 6, `MPI_Allgather` in Figure 7, `MPI_Reduce` in Figure 8, `MPI_Allreduce` in Figure 9, `MPI_Bcast` in Figure 10, and `MPI_Alltoall` in Figure 11. For all of these tests, a lower latency will lead to better performance.

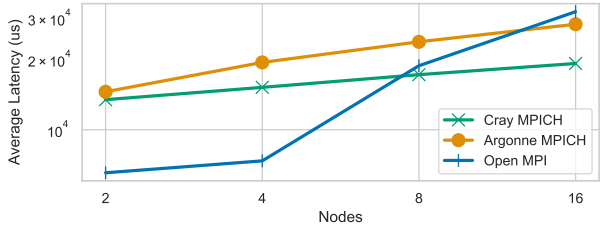

Fig. 6: MPI_Gather Strong Scaling (16MB Message Size)

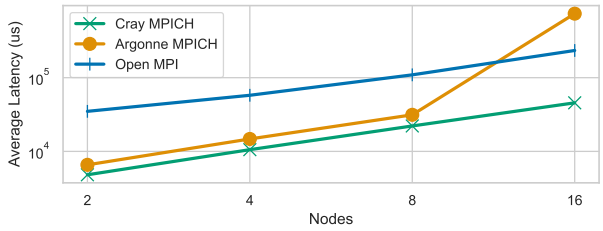

Fig. 7: MPI_Allgather Strong Scaling (16MB Message Size)

Overall, our strong scaling results indicate that Cray MPICH tends to perform the best on Slingshot-11 over a wide variety

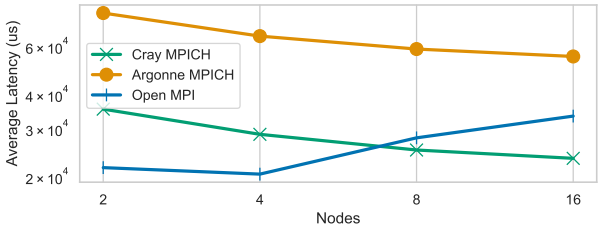

Fig. 8: MPI_Reduce Strong Scaling (16MB Message Size)

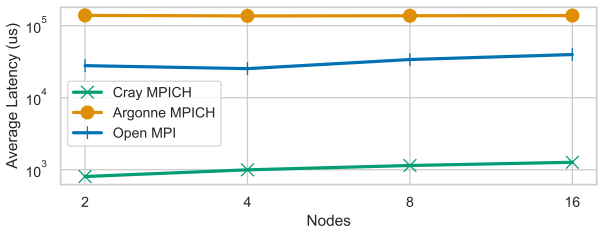

Fig. 9: MPI_Allreduce Strong Scaling (16MB Message Size)

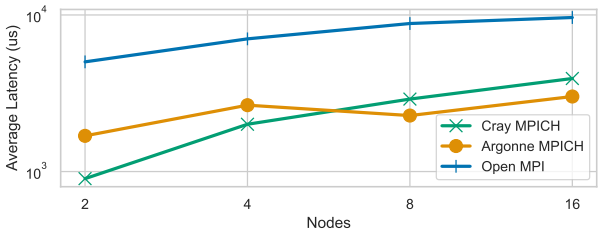

Fig. 10: MPI_Bcast Strong Scaling (16MB Message Size)

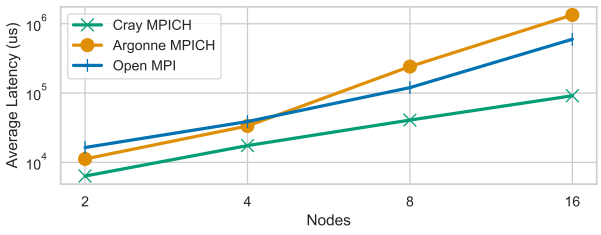

Fig. 11: MPI_Alltoall Strong Scaling (16MB Message Size)

of MPI calls and problem sizes. Figure 6 shows that Open MPI outperforms the other two implementations when using only two nodes but scales rather poorly. In Figure 7 we observe linear scaling for Cray MPICH and Open MPI, but see Argonne MPICH have a significant drop-off in performance when increasing from eight to sixteen nodes. Figure 10 shows Cray MPICH and Argonne MPICH having similar `MPI_Bcast` latencies with Argonne MPICH slightly outperforming Cray MPICH at sixteen nodes and Open MPI greatly under-performing. As shown in Figure 11, all three implementations scale linearly with Cray MPICH exhibiting the lowest latencies for each number of nodes.

2) Weak Scaling

Next, we perform several weak scaling collective tests using sixteen nodes and four processes per node. Our weak scaling is not done with the same test size across multiple number of nodes, but by changing the data being passed across the same 16 nodes as done with the strong scaling. This is done to better represent the 16 node application tests done for LAMMPS and WarpX in III-C. We again show the same MPI calls with `MPI_Gather` in Figure 12 for small data packet sizes and Figure 13 for large, `MPI_Allgather` in Figures 14 and 15, `MPI_Reduce` in Figures 16 and 17, `MPI_Allreduce` in Figures 18 and 19, `MPI_Bcast` in Figures 20 and 21, and `MPI_Alltoall` in Figures 22 and 23. Similar to the strong scaling above, Cray MPICH provides the best latency values for all but a few test instances.

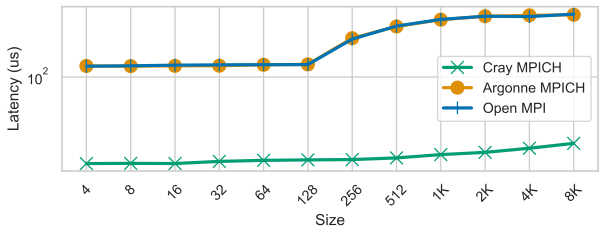

Fig. 12: MPI_Gather Weak Scaling (16 Nodes) - Small Messages

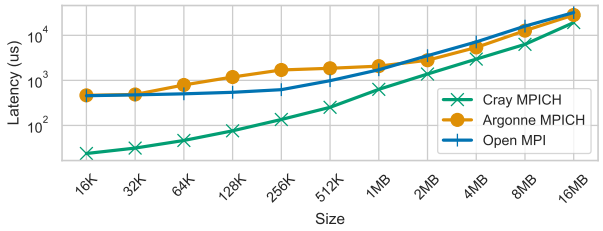

Fig. 13: MPI_Gather Weak Scaling (16 Nodes), Large Messages

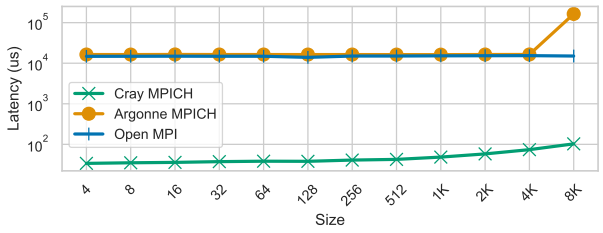

Fig. 14: MPI_Allgather Weak Scaling (16 Nodes), Small Messages

Cray MPICH outperforms Open MPI and Argonne MPICH throughout the entire tested region for `MPI_Gather` and `MPI_Allgather`, often by more than an order of magnitude. All three have step like functions at various places within these two tests. However, the different locations where the jumps occur show that this is not a hardware restriction but

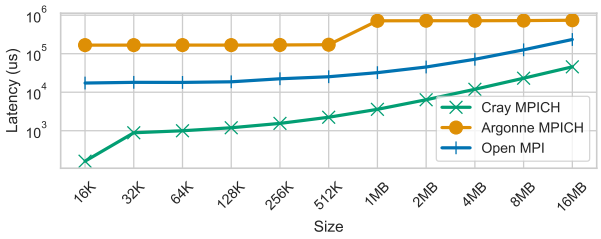

Fig. 15: MPI_Allgather Weak Scaling (16 Nodes), Large Messages

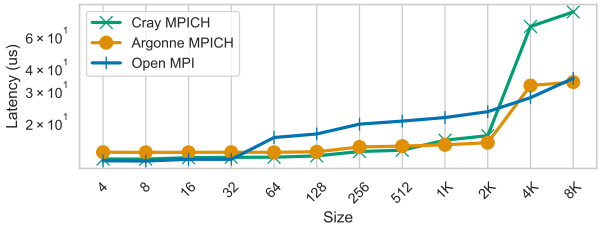

Fig. 16: MPI_Reduce Weak Scaling (16 Nodes), Small Messages

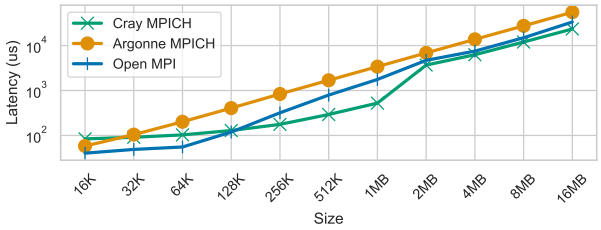

Fig. 17: MPI_Reduce Weak Scaling (16 Nodes), Large Messages

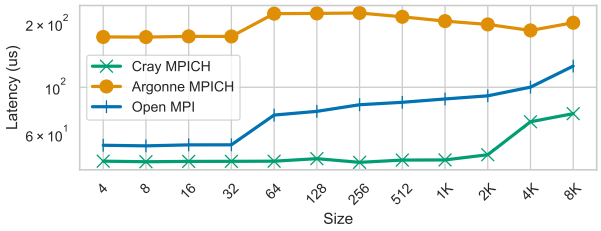

Fig. 18: MPI_Allreduce Weak Scaling (16 Nodes), Small Messages

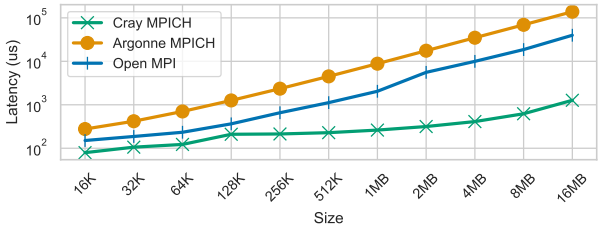

Fig. 19: MPI_Allreduce Weak Scaling (16 Nodes), Large Messages

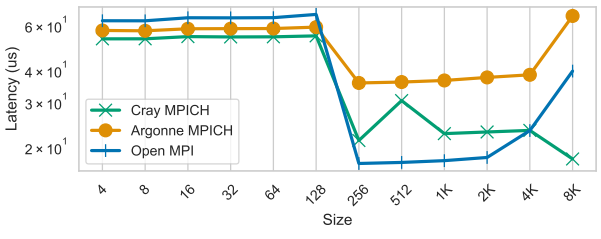

Fig. 20: MPI_Bcast Weak Scaling (16 Nodes), Small Messages

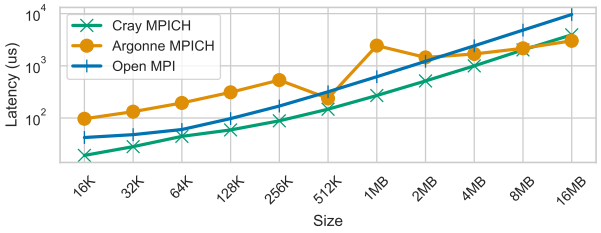

Fig. 21: MPI_Bcast Weak Scaling (16 Nodes), Large Messages

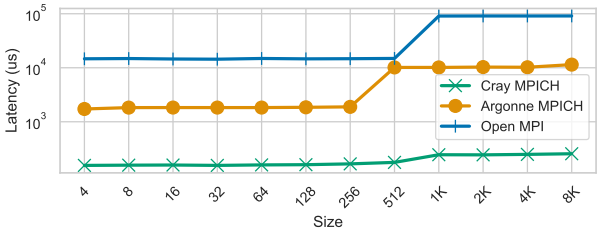

Fig. 22: MPI_Alltoall Weak Scaling (16 Nodes), Small Messages

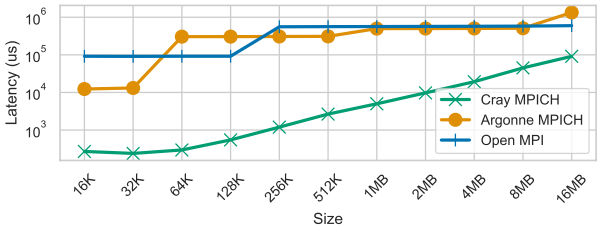

Fig. 23: MPI_Alltoall Weak Scaling (16 Nodes), Large Messages

rather differences in the MPI implementations. As shown in Figures 12 and 13 Cray MPICH outperforms Argonne MPICH and Open MPI when performing `MPI_Gather` with small messages, with the three implementations beginning to even out as the message size increases beyond 16K. This disparity is more pronounced when performing `MPI_Allgather` as evidenced by Figures 14 and 15. All three implementations show similar performance for `MPI_Reduce` on sixteen nodes across message sizes with Cray MPICH lagging behind between 2K and 16K. but overtaking the other two implementations after 128K. For `MPI_Allreduce`, we see Argonne MPICH and Open MPI scale somewhat linearly, particularly for large message sizes, but with Cray MPICH showing much

smaller latencies. Figure 20 shows all three implementations having a significant drop-off in latency when increasing from 128 byte to 256 byte messages, with Open MPI having the most profound drop-off. All three implementations having a similar change in performance may be linked to an inherent attribute of the network rather than MPI implementation. Lastly, Cray MPICH outperforms Argonne MPICH and Open MPI at `MPI_Alltoall` on sixteen nodes for all message sizes. For this routine, Argonne MPICH performs better than Open MPI with smaller messages but is later surpassed as the message size increases to 16MB.

C. Scientific Application Analysis

1) LAMMPS

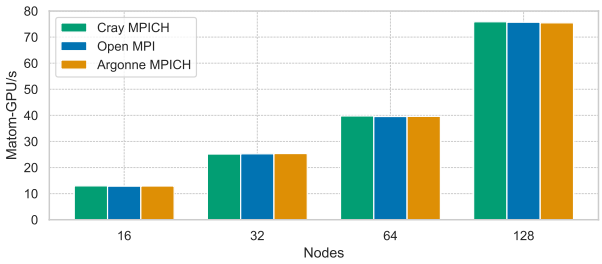

Fig. 24: LAMMPS SNAP Potential (Higher is Better)

For our LAMMPS test, we ran the Spectral Neighbor Analysis Potential (SNAP) [12]. The SNAP potential is an inter-atomic potential model that captures complex atomic interactions using bi-spectrum components of local atomic environments. A great deal of work has been done in LAMMPS to hide communication latency and minimize the overhead resulting from data transfers. The results of this work are reflected in our SNAP output, as the performance discrepancy across MPI implementations is minuscule for each number of nodes used. The similar performance across implementations shows that optimizations may be made to reduce the communication constraints at the application level to bridge the gap between the performance of MPI implementations despite their varying degrees of support for different interconnects.

2) WarpX

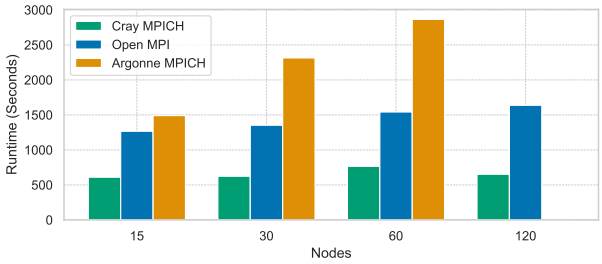

Fig. 25: WarpX KPP (Lower is Better)

The WarpX test shows significant difference in application runtime using the different MPI implementations. The Cray MPICH performs best, followed by OpenMPI and then the Argonne MPICH as shown in Figure 25. This can be attributed to the MPI routines that WarpX uses, as it uses a large number MPI sends and receives with many of the messages sizes in the range of 256 bytes to 16K, where the Cray MPICH has lower latencies than the other implementations by a large margin. Furthermore, WarpX is performing many point-to-point communications with large message sizes ranging from 1MB to 6MB, where Argonne MPICH exhibits exceedingly low communication bandwidth making it an order of magnitude slower than the other implementations. A 120 node WarpX run for Argonne MPICH is not reported as its low bandwidth for large message sizes prevented it from being able to run within a reasonable time frame.

IV. Related Work

Several studies [13] [14] have been done to compare the performance of MPI implementations and analyze the intricacies of GPU-aware MPI for varied use cases [15]. Khorassani *et al.* [13] did an early evaluation of Slingshot-11 shortly after its release comparing MVAPICH-2 and Cray MPICH on CPUs. Studies have been performed to compare the performance of pure MPI vs. hybrid MPI with OpenMP [16].

V. Future Work

In future work, we will expand on this study and address its limitations. For instance, we will examine other MPI implementations with GPU-aware support like MVAPICH-2 [17]. Another important addition will be to extend our study to other popular high-speed interconnects such as Infiniband and Omni-Path. Furthermore, we will evaluate a wider variety of "real-world" applications that cover the diverse set of communication patterns characteristic of modern HPC applications. Since MPI performance characteristics are not limited to communication latency and bandwidth, it will be beneficial to consider other metrics such as memory usage, energy consumption, startup time, etc.

VI. Conclusion

Rapid evolution in interconnect technology and GPU-computing has led to communication performance portability challenges across MPI implementations. In this work, we use several scientific applications and benchmarks to evaluate the communication performance of different GPU-aware MPI implementations including Cray MPICH, Argonne MPICH, and Open MPICH over the Slingshot-11 interconnect on the Perlmutter supercomputer. Our experimental results show that HPE's Cray MPICH outperforms the others in almost all scenarios due to its SS11 specific optimizations. As Argonne MPICH and Open MPI developers have more time to optimize their implementations for Slingshot-11, we anticipate this performance gap to shrink in the future. We believe that our study will help motivate MPI developers to support and optimize for new and upcoming interconnect technologies as well as help application developers to choose the best MPI implementation for their specific applications and compute environments.

ACKNOWLEDGEMENTS

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] R. Hempel and D. W. Walker, "The emergence of the mpi message passing standard for parallel computing," *Computer Standards & Interfaces*, vol. 21, no. 1, pp. 51–62, 1999.

[2] A. Thompson and C. Newburn. (2022) Gpudirect storage: A direct path between storage and gpu memory. [Online]. Available: https://developer.nvidia.com/blog/gpudirect-storage/

[3] O. S. U. Network-Based Computing Laboratory, "Osu micro-benchmarks 7.2," 2023, available at: https://mvapich.cse.ohio-state.edu/benchmarks/.

[4] A. P. Thompson, H. M. Aktulga, R. Berger, D. S. Bolintineanu, W. M. Brown, P. S. Crozier, P. J. in 't Veld, A. Kohlmeyer, S. G. Moore, T. D. Nguyen, R. Shan, M. J. Stevens, J. Tranchida, C. Trott, and S. J. Plimpton, "LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales," *Comp. Phys. Comm.*, vol. 271, p. 108171, 2022.

[5] L. Fedeli, A. Huebl, F. Boillod-Cerneux, T. Clark, K. Gott, C. Hillairet, S. Jaure, A. Leblanc, R. Lehe, A. Myers *et al.* (2022) Pushing the frontier in the design of laser-based electron accelerators with groundbreaking mesh-refined particle-in-cell simulations on exascale-class supercomputers. IEEE. Available at: https://github.com/ECP-WarpX/WarpX.

[6] H. P. Enterprise, "Hpe slingshot interconnect," 2023, available at: https://www.hpe.com/us/en/compute/hpc/slingshot-interconnect.html.

[7] D. De Sensi *et al.*, "An in-depth analysis of the slingshot interconnect," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020, pp. 1–14. [Online]. Available: https://doi.org/10.1109/SC41405.2020.00039

[8] N. Documentation, "Cray mpich," 2023, available at: https://docs.nersc.gov/programming/mpi/cray-mpich/.

[9] N. T. Karonis, B. Toonen, and I. Foster, "Mpich-g2: A grid-enabled implementation of the message passing interface," *arXiv:cs/0206040v2*, June 2002, available at: https://arxiv.org/abs/cs/0206040v2. [Online]. Available: https://arxiv.org/abs/cs/0206040v2

[10] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, September 2004. [Online]. Available: https://www.open-mpi.org/papers/euro-pvmmpi-2004-overview/

[11] W. Zhang, A. Almgren, V. Beckner, J. Bell, J. Blaschke, C. Chan, M. Day, B. Friesen, K. Gott, D. Graves, M. Katz, A. Myers, T. Nguyen, A. Nonaka, M. Rosso, S. Williams, and M. Zingale, "AMReX: a framework for block-structured adaptive mesh refinement," *Journal of Open Source Software*, vol. 4, no. 37, p. 1370, May 2019. [Online]. Available: https://doi.org/10.21105/joss.01370

[12] R. Gayatri, S. Moore, E. Weinberg, N. Lubbers, S. Anderson, J. Deslippe, D. Perez, and A. P. Thompson, "Rapid exploration of optimization strategies on advanced architectures using testsnap and lammps," *arXiv preprint arXiv:2011.12875*, 2020, available at: https://arxiv.org/abs/2011.12875v1.

[13] K. S. Khorassani *et al.*, "High performance mpi over the slingshot interconnect," *Journal of Computer Science and Technology*, vol. 38, no. 1, pp. 128–145, 2023. [Online]. Available: https://doi.org/10.1007/s11390-023-2907-5

[14] H. Wang *et al.*, "Gpu-aware mpi on rdma-enabled clusters: Design, implementation and evaluation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2595–2605, 2014. [Online]. Available: https://doi.org/10.1109/TPDS.2013.222

[15] M. Aissa *et al.*, "Toward a gpu-aware comparison of explicit and implicit cfd simulations on structured meshes," *Computers & Mathematics with Applications (1987)*, vol. 74, no. 1, pp. 201–217, 2017. [Online]. Available: https://doi.org/10.1016/j.camwa.2017.03.003

[16] N. Drosinos and N. Koziris, "Performance comparison of pure mpi vs hybrid mpi-openmp parallelization models on smp clusters," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings*. IEEE, 2004, p. 15. [Online]. Available: https://doi.org/10.1109/IPDPS.2004.1302919

[17] D. K. Panda, H. Subramoni, C.-H. Chu, and M. Bayatpour, "The mvapich project: Transforming research into high-performance mpi library for hpc community," *Journal of Computational Science*, vol. 52, p. 101208, 2021, case Studies in Translational Computer Science. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877750320305093