MPI Advance : Open-Source Message Passing Optimizations

AMANDA BIENZ*, Dept. of Computer Science
Univ. of New Mexico, USA
DEREK SCHAFER*, Center for Advanced Research Computing
Univ. of New Mexico, USA
ANTHONY SKJELLUM*, Dept. of Computer Science
Tennessee Technological University, USA

The large variety of production implementations of the message passing interface (MPI) each provide unique and varying underlying algorithms. Each emerging supercomputer supports one or a small number of system MPI installations, tuned for the given architecture. Performance varies with MPI version, but application programmers are typically unable to achieve optimal performance with local MPI installations and therefore rely on whichever implementation is provided as a system install. This paper presents MPI Advance, a collection of libraries that sit on top of MPI, optimizing the underlying performance of any existing MPI library. The libraries provide optimizations for collectives, neighborhood collectives, partitioned communication, and GPU-aware communication.

CCS Concepts: • **Networks** → **Network performance evaluation**; **Network performance modeling**; • **Computing methodologies** → Massively parallel algorithms.

ACM Reference Format:
Amanda Bienz, Derek Schafer, and Anthony Skjellum. 2023. MPI Advance : Open-Source Message Passing Optimizations. 1, 1 (September 2023), 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

A large variety of message-passing interface (MPI) libraries are currently in production, including OpenMPI, MPICH, MVAPICH, and various proprietary implementations. Each emerging supercomputer provides a system install of one or a small number of MPI implementations, tuned to obtain optimal performance for a given architecture. While each MPI implementation provides the standard API, underlying implementations vary drastically. As a result, the performance of parallel applications is dependent on not only the architecture on which they are run but also the implementations within the available system MPI install. While additional versions of MPI can be installed through package managers, such as Spack, performance will typically be subpar in comparison to tuned system installations. In this paper, we introduce MPI Advance[1], a collection of lightweight libraries that sit on top of MPI, providing advanced algorithms and new MPI features while also leveraging the tuned performance of the system MPI.

There are many benefits to lightweight libraries that sit on top of MPI, such as MPI Advance. The first is that the simplicity of this approach allows users to experiment with research MPI extensions without having to edit production

*All authors contributed equally to this research.
[1]https://github.com/mpi-advance

Manuscript submitted to ACM

MPI releases, allowing libraries to be portably tested across various MPI implementations and computer architectures more directly. Such a design also allows for the creation of libraries that applications can utilize to use communication optimizations, such as emerging algorithms for collective operations and locality-aware aggregation [3, 4, 10]. MPI Advance also provides the opportunity for users to add new MPI methods or optimizations that are not yet available within the MPI standard, allowing for testing within a variety of applications to gather evidence for whether new ideas should be added to the standard. Finally, the libraries allow for implementations of new additions to the standard, such as partitioned communication [7], to be made available to application users before system MPI installs have been updated. The end goal of most MPI Advance libraries will be that the learning associated with creating the library can make the transition of the library into a production MPI release more streamlined.

The remainder of this paper describes MPI Advance in detail. Section 2 outlines what libraries are currently included within MPI Advance, along with the specific rationale for being included. Benefits and preliminary performance results of these libraries are then described in Section 3. Finally, Section 4 provides concluding remarks.

2 MPI ADVANCE

MPI Advance provides a framework for distributing communication optimizations, including optimizations to collective algorithms and neighborhood collectives. The codebase also provides access to algorithms that have recently been added to the MPI standard, but are not yet provided in many system MPI installations, such as partitioned communication and persistent collective operations. MPI Advance also provides a library for GPU-Aware communication for heterogeneous architectures, providing the choice for utilizing GPUDirect when available, copying data to a single CPU, or multithreading data copied to the CPU to utilize all available cores. Finally, all MPI Advance libraries utilize the MPIX prefix in the user-facing APIs [13, 14].

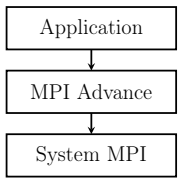

Fig. 1. Dependencies

MPI Advance sits between the application and system MPI installation, as exemplified in Figure 1. The library utilizes the MPIX-extensions, allowing for all applications to access MPI Advance optimizations without conflict. MPI Advance then relies on the system MPI implementation for underlying message passing.

2.1 Collective Optimizations

Collective algorithms have been optimized over multiple decades to minimize message count for small data sizes and bytes transported for larger messages. In more recent years, architecture-aware algorithms further optimized many collectives by distinguishing between inter- and intra- node communication, reducing message count and size transported through the network. Further optimizations, such as topology-aware collectives, gain additional performance by minimizing hop count. MPI Advance provides an interface for a variety of these collective implementations in which the various algorithms are implemented on top of MPI, typically calling point-to-point communication within the underlying MPI implementation.

Currently, the collectives in this MPI Advance library select a default algorithm, but make all implementations publicly available so that users can select a non-default algorithm. Future work includes research into a more sophisticated selection process to help choose optimal collective algorithms for a given architecture and underlying MPI implementation. Listings 1 and 2 show how this library can be used to replace an existing all-to-allv operation inside an MPI application.

Listing 1. Standard

```
MPI_Alltoallv(sendbuf, sendcounts,
    sdispls, sendtype, recvbuf,
    recvcounts, rdispls, recvtype,
    comm);
```

Listing 2. MPI Advance

```
MPIX_Comm* xcomm;
MPIX_Comm_init(&xcomm, comm);
MPIX_Alltoallv(sendbuf, sendcounts, sdispls,
    sendtype, recvbuf, recvcounts, rdispls,
    recvtype, xcomm);
```

2.2 Persistent Neighborhood Collective Optimizations

Neighborhood collectives allow MPI to optimize sparse communication, in which each process communicates with a subset of other processes. Persistent versions of neighborhood collectives were added to the MPI 4 standard, allowing for all setup costs to be incurred only once in the initialization method. While standard implementations of neighbor collectives typically consist of simply wrapping point-to-point communication, the persistent API allows for optimizations of the underlying communication, such as locality-aware aggregation. MPI Advance provides a library featuring locality-aware optimizations to persistent neighborhood collectives.

Listing 3. Standard

```
MPI_Comm comm;
MPI_Dist_graph_create_adjacent(
    comm, n_recvs, recv_procs,
    recv_weights, n_sends,
    send_procs, send_weights,
    mpi_info, reorder, &comm);
MPI_Neighbor_alltoallv(sendbuf,
    sendcounts, sdispls, sendtype,
    recvbuf, recvcounts, rdispls,
    recvtype, comm);
```

Listing 4. MPI Advance

```
MPIX_Comm xcomm;
MPIX_Request* xrequest;
MPIX_Dist_graph_create_adjacent(comm,
    n_recvs, recv_procs, recv_weights,
    n_sends, send_procs, send_weights,
    mpi_info, reorder, &xcomm);
MPIX_Neighbor_alltoallv_init(sendbuf,
    sendcounts, sdispls, sendtype, recvbuf,
    recvcounts, rdispls, recvtype, xcomm,
    info, &xrequest);
MPIX_Start(xrequest);
MPIX_Wait(xrequest, MPI_STATUS_IGNORE);
```

Persistent neighborhood collectives within this library can be used similarly to the collective operations previously described. The neighborhood collective in Listing 3 only needs to be replaced with the persistent MPIX version in Listing 4. Currently, there is an additional locality-aware extension to the neighborhood collective that requires additional data, namely unique indices for all values to be sent and received. These unique values allow for the neighborhood collective to eliminate a single value from being sent between a set of nodes multiple times.

2.3 Partitioned Communication

Partitioned collective communication [8, 11], is a new, channelized approach to point-to-point communication added in the MPI-4 standard. Partitioned point-to-point communication establishes a single match between a sender and receiver at initialization, and supports the marking of portions of buffers (partitions), that can be transferred before the entire message is complete. In this way, on the send side, partitioned sends allow so-called early-bird communication. On the receive-side, partitions of the same or different sizes can be accessed as complete, prior to completion of the whole message. In this paradigm, two-sided operations enable one-sided implementation internally, and support overlap of communication and computation in strong-progress implementations. In situations where partitions are computed in

parallel, such as in fork-join parallelism with OpenMP or CUDA kernels, partitioned communication helps hide the load imbalance of such computations and ensures that the computations and transfers have a good opportunity to overlap.

MPIPCL [7] is a component of MPI Advance that supports the complete semantics of MPI-4 partitioned point-to-point operations with a reasonably performant interface. The architecture of MPICL introduces a progress thread in order to move data asynchronously even when the underlying MPI implementation does not provide such a guarantee. The semantics of initialization, initiation, and the Pready/Parrived partition operations are fully supported. Simple partitioning strategies are also supported. Significantly, MPICL can port on top of any MPI-3.0 compliant implementation that offers `MPI_Thread_multiple` modes of execution. The audience for these APIs are early adopters of MPI+X modes of operation and implementers of partitioned operations in production MPI middleware. Additional extensions involving collective communication are being proposed for MPI-5 [9]. Similar to the point-to-point operations, a layered-library implementation of these collective APIs is being developed for MPIPCL [1] and will soon be available for public use.

2.4 Heterogeneous Architectures

Emerging heterogeneous architectures often require applications to communicate data between GPU memories. There are many different paths of data transfer, including using GPUDirect to move data directly from the GPU memory to the NIC, copying all data to a CPU before performing standard MPI communication between GPUs, copying portions of the data to each of the available CPU cores to distribute communication, or some combination of these protocols. MPI Advance supports CUDA-aware and ROCM-aware communication, providing implementations for each of the various protocols mentioned above. To utilize GPU-aware communication, this MPI Advance library must be compiled with either the CUDA or HIP flags. Users can then call MPI operations, such as collectives and neighborhood collectives, passing either CPU or GPU memory. While the libraries automatically select default implementations, all existing implementations are publicly available for the user to switch among them.

3 BENEFITS AND RESULTS

MPI Advance provides a lightweight interface that allows users to select specific implementations of methods within the MPI standard, create additional optimized implementations for existing methods, utilize new methods not yet within the standard or common installations of MPI, and perform MPI research without editing production MPI libraries.

Many of the optimizations within MPI Advance have been published in other papers, including locality-aware neighborhood collectives [6], locality-aware all-gather operations [2], and optimized all-to-allv implementations [12]. Furthermore, MPI Advance provides options for GPUDirect, copy-to-CPU, and copy-to-many-CPU approaches, as previously analyzed on Lassen and Summit, Power9 systems at LLNL and ORNL, respectively [5].

MPIPCL has been similarly benchmarked. Comparisons against other MPI point-to-point operations [15] find that with only one partition, MPIPCL is no worse than base point-to-point operations. The second test was against an internal, RMA-based implementation of the partitioned point-to-point APIs [7], finding that MPIPCL has similar performance, noting that MPIPCL may fall behind the internal implementation as it leverages more internal optimizations.

4 CONCLUSIONS

Establishing community best practices, acceptance, and use of new and revised MPI features is crucial to maintaining its value as a key parallel programming model and continuing to deliver on its promise of performance-portability in the Exascale era. To that end, MPI Advance provides access to new standardized features in MPI on existing production systems that are not updated, and it provides access to experimental and future standard features long before they

appear in the MPI standard or production implementations. This set of capabilities helps speed up adoption and provides early feedback to the designers of new features before new editions of the MPI Standard are finalized.

ACKNOWLEDGMENTS

This work was performed with partial support from the National Science Foundation under Grants Nos. CCF-2151022, CCF-1918987, CCF-1562306, CCF-1822191, CCF-1821431, OAC-1923980, OAC-1549812, OAC-1925603, OAC-2201497, and CCF-2151020 and the U.S. Department of Energy's National Nuclear Security Administration (NNSA) under the Predictive Science Academic Alliance Program (PSAAP-III), Award DE-NA0003966. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF and the U.S. Department of Energy's NNSA.

REFERENCES

[1] Muzakhir Amanzholov. 2022. *Functional Implementation of Partitioned Collective Communication Primitives.* Master's thesis. Tennessee Technological University, Cookeville TN.

[2] Amanda Bienz, Shreeman Gautam, and Amun Kharel. 2022. A Locality-Aware Bruck Allgather. In *Proceedings of the 29th European MPI Users' Group Meeting* (Chattanooga, TN, USA) *(EuroMPI/USA'22)*. Association for Computing Machinery, New York, NY, USA, 18–26. https://doi.org/10.1145/3555819.3555825

[3] Amanda Bienz, William D. Gropp, and Luke N. Olson. 2020. Reducing communication in algebraic multigrid with multi-step node aware communication. *The International Journal of High Performance Computing Applications* 34, 5 (2020), 547–561. https://doi.org/10.1177/1094342020925535 arXiv:https://doi.org/10.1177/1094342020925535

[4] Amanda Bienz, Luke N. Olson, and William D. Gropp. 2019. Node aware sparse matrix-vector multiplication. *J. Parallel and Distrib. Comput.* 130 (2019), 166 – 178. https://doi.org/10.1016/j.jpdc.2019.03.016

[5] Amanda Bienz, Luke N. Olson, William D. Gropp, and Shelby Lockhart. 2021. Modeling Data Movement Performance on Heterogeneous Architectures. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)* (Waltham, MA, USA). IEEE, 1–7. https://doi.org/10.1109/HPEC49654.2021.9622742

[6] Gerald Collom, Rui Peng Li, and Amanda Bienz. 2023. Optimizing Irregular Communication with Neighborhood Collectives and Locality-Aware Parallelism. arXiv:2306.01876 [cs.DC]

[7] Matthew G.F. Dosanjh, Andrew Worley, Derek Schafer, Prema Soundararajan, Sheikh Ghafoor, Anthony Skjellum, Purushotham V. Bangalore, and Ryan E. Grant. 2021. Implementation and evaluation of MPI 4.0 partitioned communication libraries. *Parallel Comput.* 108 (2021), 102827. https://doi.org/10.1016/j.parco.2021.102827

[8] Ryan E. Grant, Matthew G. F. Dosanjh, Michael J. Levenhagen, Ron Brightwell, and Anthony Skjellum. 2019. Finepoints: Partitioned Multithreaded MPI Communication. In *High Performance Computing - 34th International Conference, ISC High Performance 2019, Frankfurt/Main, Germany, June 16-20, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11501)*, Michèle Weiland, Guido Juckeland, Carsten Trinitis, and Ponnuswamy Sadayappan (Eds.). Springer, Frankfurt, Germany, 330–350. https://doi.org/10.1007/978-3-030-20656-7_17

[9] Daniel J. Holmes, Anthony Skjellum, Julien Jaeger, Ryan E. Grant, Purushotham V. Bangalore, Matthew G.F. Dosanjh, Amanda Bienz, and Derek Schafer. 2021. Partitioned Collective Communication. In *2021 Workshop on Exascale MPI (ExaMPI)*. IEEE, 9–17. https://doi.org/10.1109/ExaMPI54564.2021.00007

[10] Shelby Lockhart, Amanda Bienz, William Gropp, and Luke Olson. 2023. Performance Analysis and Optimal Node-Aware Communication for Enlarged Conjugate Gradient Methods. *ACM Trans. Parallel Comput.* 10, 1, Article 2 (mar 2023), 25 pages. https://doi.org/10.1145/3580003

[11] MPI Forum. 2020. *MPI: A Message-Passing Interface 4.0 Standard.* Technical Report. Univ. of Tennessee, Knoxville, TN, USA.

[12] Evelyn Namugwanya, Amanda Bienz, Derek Schafer, and Anthony Skjellum. 2023. Collective-Optimized FFTs. arXiv:2306.16589 [cs.MS]

[13] Anthony Skjellum, Nathan E Doss, and Kishore Viswanathan. 1994. Inter-communicator extensions to MPI in the MPIX (MPI eXtension) Library.

[14] Anthony Skjellum, Nathan E Doss, Kishore Viswanathan, Aswini Chowdappa, and Purushotham V Bangalore. 1994. Extending the message passing interface (MPI). In *Proceedings Scalable Parallel Libraries Conference.* IEEE, Mississippi State, MS, USA, 106–118.

[15] Andrew Worley, Prema Prema Soundararajan, Derek Schafer, Purushotham Bangalore, Ryan Grant, Matthew Dosanjh, Anthony Skjellum, and Sheikh Ghafoor. 2021. Design of a Portable Implementation of Partitioned Point-to-Point Communication Primitives. In *50th International Conference on Parallel Processing Workshop* (Lemont, IL, USA) *(ICPP Workshops '21)*. Association for Computing Machinery, New York, NY, USA, Article 35, 11 pages. https://doi.org/10.1145/3458744.3474046