Extended Abstract: Taking Open MPI to New Frontiers

Amir Shehata
Thomas Naughton
David Bernholdt
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA

Howard Pritchard
Los Alamos National Laboratory
Los Alamos, New Mexico, USA

KEYWORDS

Message Passing Interface, High Performance Computing, libfabric, Open MPI, Slingshot

1 INTRODUCTION

The new exascale systems at the U.S. Department of Energy (DoE) laboratories include a new network interconnect from Cray HPE. *Frontier* at the Oak Ridge Leadership Computing Facility (OLCF) is the first DoE exascale system that includes the new Slingshot 11 network. This same interconnect is used in *Aurora* at Argonne Leadership Computing Facility (ALCF) and *Perlmutter* at NERSC. As such, the support of Slingshot 11 is an important capability to meet the needs of exascale applications.

This poster highlights the design and development of infrastructure to enable Open MPI to efficiently support the new Slingshot 11 platforms. The focus of the poster is on enhancements for intra-node and inter-node communication that uses the libfabric shared memory (SHM) and Slingshot (CXI) providers.

2 OVERVIEW

The goal of this work is to provide an alternative MPI using Open MPI that provides comparable performance to the vendor provided MPI (i.e., Cray MPI). Support for alternative MPI implementations are important on large-scale systems because they offer users more choices and can be helpful to work around problems or experiment with new features.

2.1 Technical Landscape

The software interface to the new Cray HPE Slingshot 11 network is based on the libfabric communication library. A new Slingshot 11 provider (CXI) has been added, which provides a user-level library interface to the Cassini network devices. In the context of Open MPI, there are a few different options for using libfabric to access the SS11 network. We have considered three pathways (Figure 1):

(1) MTL (Matching Transport Layer) path – use libfabric tagged message interface
(2) BTL (Byte Transfer Layer) path – use MPI for tag matching & high-level logic & libfabric for byte transfer
(3) UCX (Unified Communication X) path – use UCX and integrate libfabric under the UCX API

Ultimately, we chose to focus on the Open MPI MTL pathway, which is described in this poster. This approach gives us the following advantages:

- MTL path has been optimized to work with OFI,
- Leverages rich set of functionality in CXI provider,
- Improved support for libfabric provider composability.

3 LINKX

3.1 Design Overview

The Open MPI MTL framework supports selection of exactly one component. This limits MTL path from being able to handle both inter- and intra-node communication efficiently with the single libfabric component. On Frontier, it will select the CXI libfabric provider and use it for both local and remote node communication. However, the CXI provider does not support a shared memory mechanisms, which results in degraded performance.

The design chosen to support Open MPI on Frontier centers around developing a new libfabric provider, LINKx. The main purpose of this provider is to link multiple providers into a single "link",

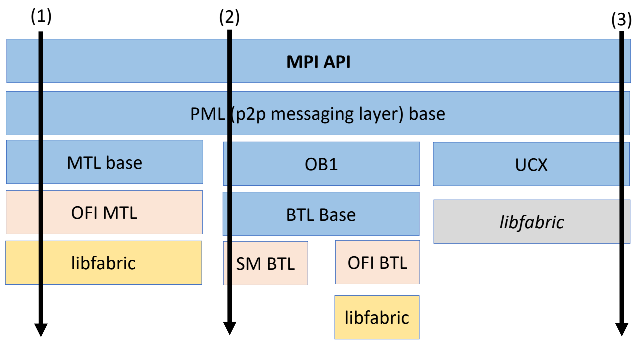


Figure 1: Open MPI Software Architecture with indication of the three potential solution paths for using the CXI provider.

Notice: This manuscript has been authored in part by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

EuroMPI'23, Sep. 11-13, 2023, Bristol, UK
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

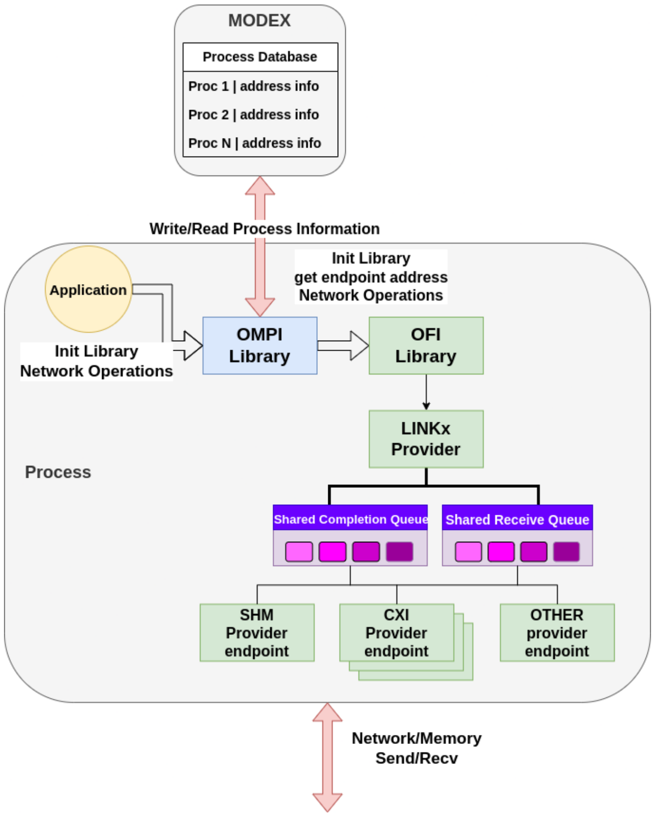


Figure 2: Illustration of OFI libfabric LINKx provider joining SHM and CXI core providers for use by Open MPI library, which uses new shared queue capability managed by LINKx.

which the application layer can use without direct knowledge of the underlying communication providers.

On Frontier, LINKx is used to link the SHM provider and the CXI provider (Figure 2). The MPI layer, therefore, becomes agnostic to which underlying communication mechanism is used. For on-node processes LINKx will use the SHM provider and for off-node processes it will use the CXI provider.

Pushing this functionality in the libfabric library as opposed to keeping it in the MPI library has the advantage of having this functionality available to other applications and/or middle-ware software that uses libfabric directly.

3.2 Flow

The following outlines the flow of steps involved when using the LINKx provider:

(1) Call `fi_getinfo()` which returns a list of `fi_info` structures; each `fi_info` describes one available communication method. Libfabric orders the list from most-to-least performant.
(2) The LINKx `fi_info` structure in the list links SHM with CXI.
(3) Application selects the LINKx `fi_info` structure and initializes using standard APIs, i.e., `fi_fabric()`, `fi_domain()`, `fi_endpoint()`
(4) LINKx builds structures to track the different SHM and CXI core providers that are part of the link.
(5) Open MPI requests the address of the LINKx provider.
(6) LINKx concatenates the addresses of the core providers.
(7) Open MPI then publishes the the local address info to MODEX.
(8) Open MPI reads all peer addresses from MODEX and inserts them into LINKx address vector table.
(9) LINKx parses the addresses and inserts them into the corresponding core provider in the link.
(10) Open MPI uses the libfabric communication APIs, e.g., `fi_tsenddata()`.
(11) LINKx examines the peer locality and determines the appropriate provider to use for communication.

3.3 Shared Peer Structures

A Peer provider design has been adopted, which allows LINKx to share its data structures with the core providers it is linking, namely CXI and SHM. The two data structures that adopt this design are the completion queue, used to post completion events to the application, and the receive queue, which is used by the application to post receive requests.

Both of these data structures must be shared in this manner to abstract the details of the link from the application. Furthermore, the receive queue must be shared to avoid race conditions where the core providers might use the same receive request to satisfy an incoming message. The core providers pull requests off the shared receive queue, which LINKx manages to prevent the aforementioned race condition. Sharing the receive queue in this manner will preclude the usage of hardware capabilities, such as hardware tag-matching, since that will break the design paradigm, leading to potential data corruption. As shown in the Figure 3, we measured CXI performance with and without hardware tag-matching, and it appears like there is no significant performance difference. However, we have not compared the CPU usage between both runs and only tested with synthetic benchmarks. In future work, we intend to expand the Shared Peer Structures design to include Address Vector tables and Memory Registration.

3.4 Improvements

To improve shared memory device-to-device, i.e., GPU, performance we fully implemented ROCm HSA APIs and added asynchronous ROCm IPC support in the SHM provider. This included additions for IPC handle caching to reduce memory attach/detach overheads. As shown in Figure 4, these improvements resulted in point-to-point memory performance on par with Cray MPICH.

To improve host-to-host memory performance on the Cray XE systems we added support for XPMEM. We leveraged XPMEM support for mapping remote process memory into the local address space to support copying from remote buffers into local device buffers.

The inter-process communication locking strategy was improved to avoid unnecessary serialization. This improved workloads like `MPI_Alltoall()` as shown in Figure 5.

Lastly, the network interface selection was improved to incorporate node topology information for optimal process to interface mapping (Figure 5).

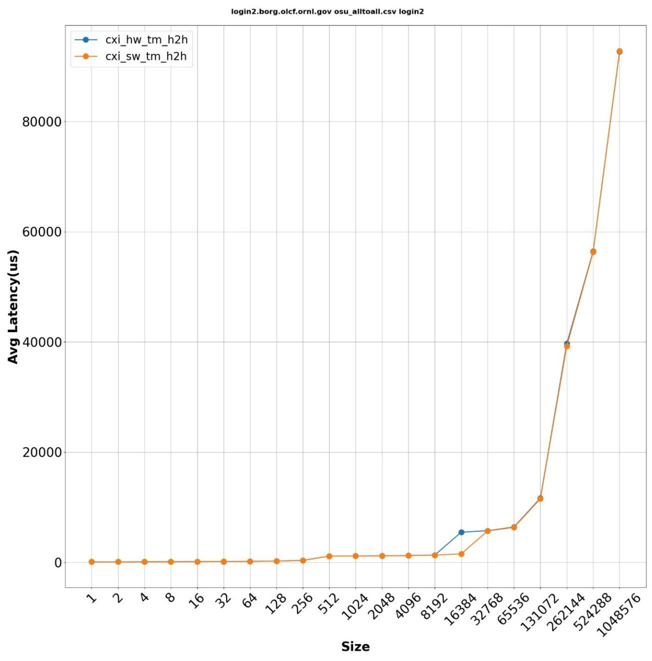


Figure 3: Collective latency CPU-to-CPU memory with and without hardware tag matching *osu-alltoall H2H* ($np = 512$ with $ppn = 8$ on $nodes = 64$)

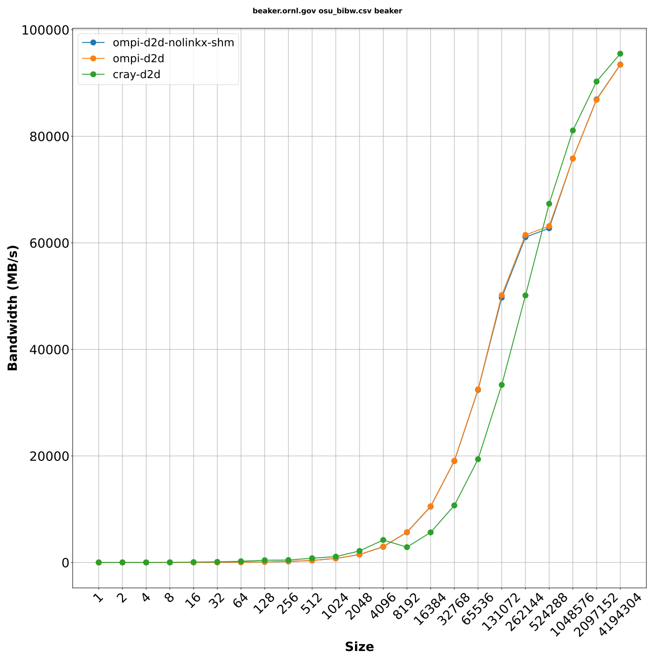


Figure 4: Point-to-point bi-directional bandwidth with GPU-to-GPU memory (intra-node) *osu-bibw D2D* ($np = 2$ with $ppn = 2$ on $node = 1$)

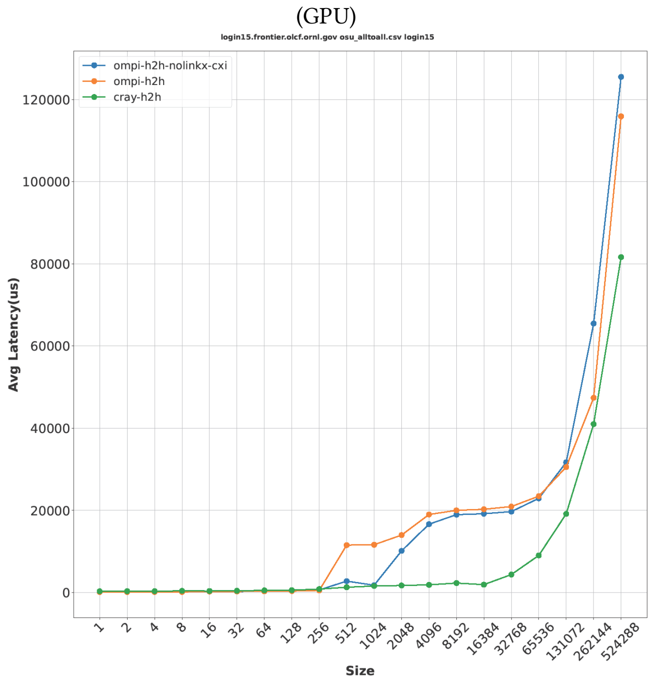


Figure 5: Collective latency CPU-to-CPU memory *osu-alltoall H2H* ($np = 1024$ with $ppn = 8$ on $nodes = 128$)

4 CONCLUSION

This poster highlights recent work to enable Open MPI to efficiently support the new Slingshot 11 platforms that are emerging at several U.S. Department of Energy supercomputing facilities. The libfabric changes include support for joining multiple providers (e.g., CXI and SHM) into a single "link"', which the application layer can use without direct knowledge of the underlying communication details. This work is beneficial to other libfabric enabled middleware software projects that seek to support exascale application on the emerging exascale systems.

This poster highlights architectural and design choices for effective use of shared memory and inter-node communication involving both host and device memory. We include initial performance results for message passing interface benchmarks using SS11 on systems running at OLCF.

Note: Please see poster for additional details & performance graphs.

ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725. Howard Pritchard acknowledges support by the National Nuclear Security Administration. Los Alamos National Laboratory is operated by Triad National Security, LLC for the U.S. Department of Energy under contract 89233218CNA000001. LA-UR-23-23988. This research was partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Taking Open MPI to New Frontiers

Amir Shehata [1], Thomas Naughton[1], David Bernholdt[1], Howard Pritchard[2]

[1] Oak Ridge National Laboratory
[2] Los Alamos National Laboratory

Problem Statement & Goals

Problems addressed:

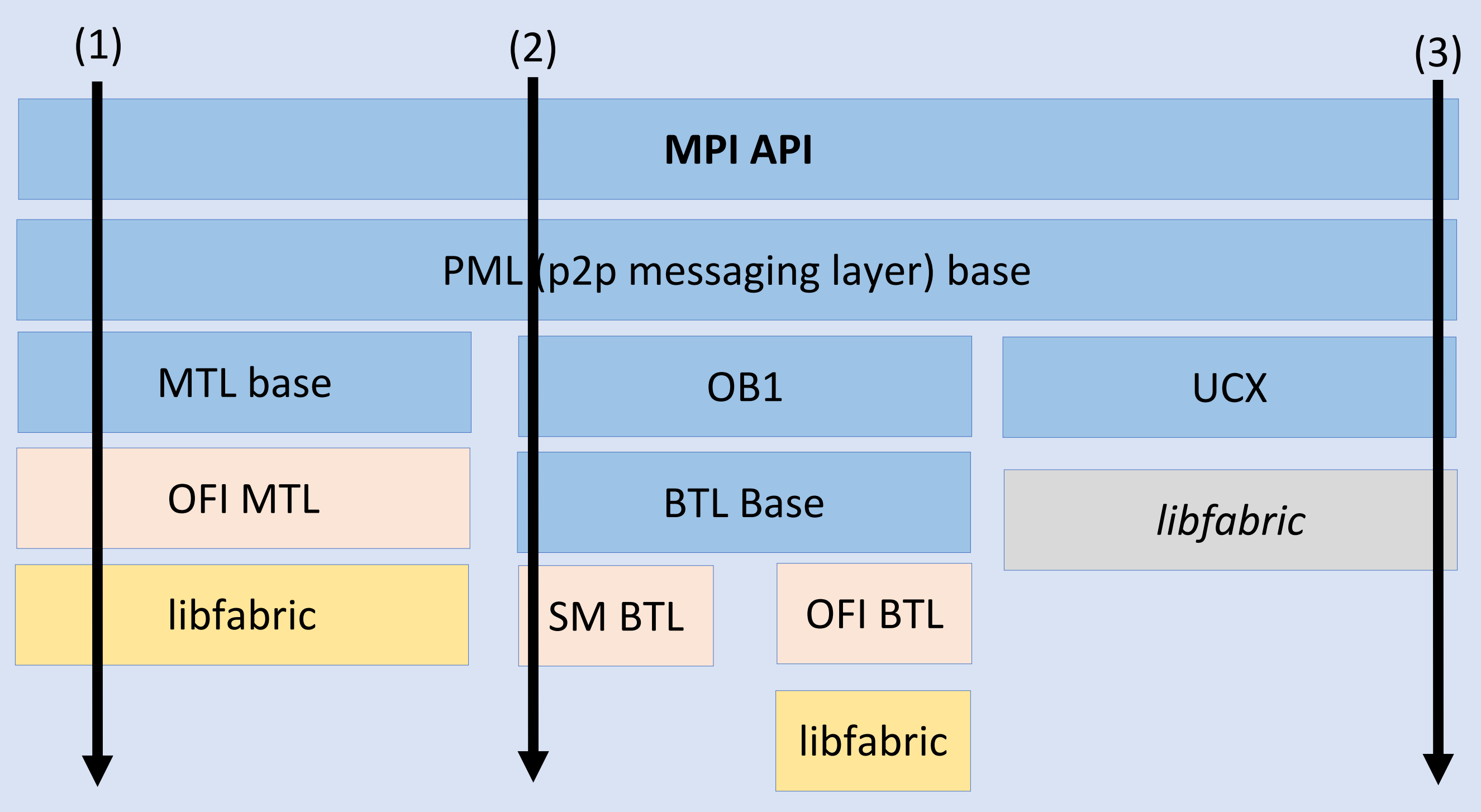
- Vendor only provides Cray MPI on Frontier
- Users need more choice of MPI implementations (work around problems, try out new features)

Goals:

- Provide an alternative MPI using Open MPI that provides comparable performance to Cray MPI

Technical Landscape

- Cray support Slingshot 11 via a new CXI libfabric provider
- Three potential solutions to use the CXI provider
  1. Open MPI MTL path - use libfabric tagged message interface
  2. Open MPI BTL path - use MPI for tag matching & high-level logic & libfabric for byte transfer
  3. Open MPI UCX path - use UCX and integrate libfabric under the UCX API
- Open MPI MTL option was selected
  - This solution represents the most logical path forward, as it makes full use of the CXI provider's functionality; also allows us to improve libfabric in its totality to support provider composability.



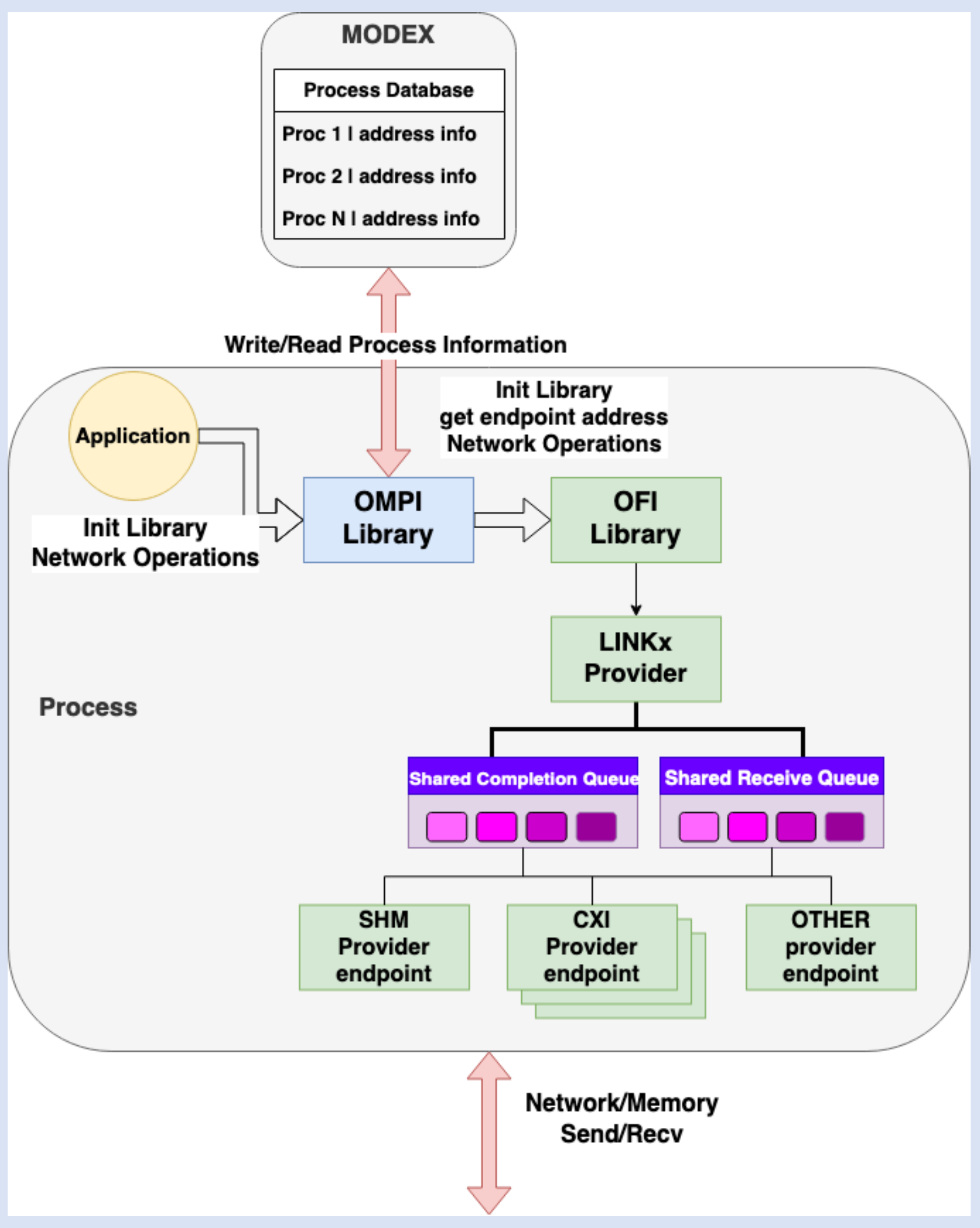
LINKx Provider

- New provider designed to link multiple providers
- Allows Open MPI to use libfabric for both local and remote communication
  - Endpoint selection based on peer locality
- LINKx shares both completion & receive queues
  - Reduce communication & memory overhead
  - Disable HW tag matching for data consistency

Example of Open MPI initialization with LINKx

1. Initialize libfabric to get LINKx provider with SHM+CXI
2. Application does typical libfabric setup for provider
3. LINKx builds structures to track linked providers
4. Open MPI MODEX: Before exchange, LINKx concatenates all addresses in link and publishes
5. Open MPI MODEX: After exchange, Open MPI reads all addresses, LINKx parses & sets up linked providers
6. Open MPI uses libfabric APIs to communicate with peers. At runtime, LINKx examines peer & selects best provider based on locality.

Solution Overview



Improvements

Libfabric Shared Memory (SHM) Provider:

To support all MPI use cases the following SHM provider features have been added

- Full support for ROCM HSA APIs
- Added Asynchronous ROCM IPC Support
- Added IPC Caching mechanism
- Added XPMEM Support
  - XPMEM allows mapping remote process memory space locally. This provides an efficient method of sharing memory
- Support H2D via XPMEM, since XPMEM maps remote process memory locally, it can then be copied directly into Device memory
- SHM locking improvements
  - SHM provider locking was very coarse, causing serialization between processes
    - Moved to a lockless strategy to minimize serialization

Network Interface Selection:

- Use distance topology information for optimal process to interface binding

Testing Environment

- "Frontier" system at OLCF
- Set the HIP_VISIBLE_DEVICES to nearest GPU(s)
- OMPI binds each process to the nearest NIC
- Alpha release

```
    module use /sw/frontier/ums/ums024/cce/15.0.0/modules
    module load openmpi
    mpirun --bind-to core --map-by ppr:1:l3cache \
           --np $SLURM_NTASKS ./application
```

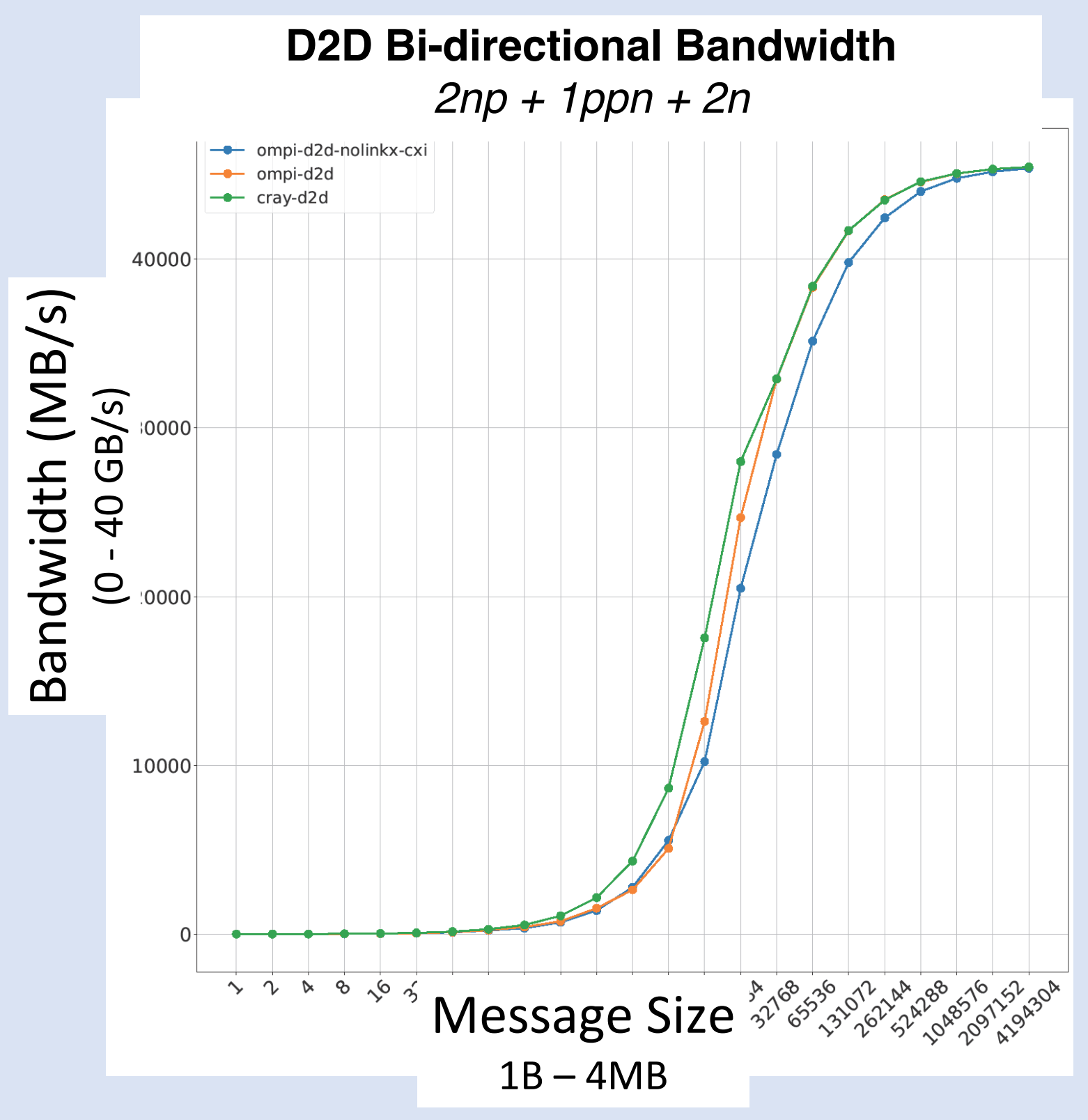
Status

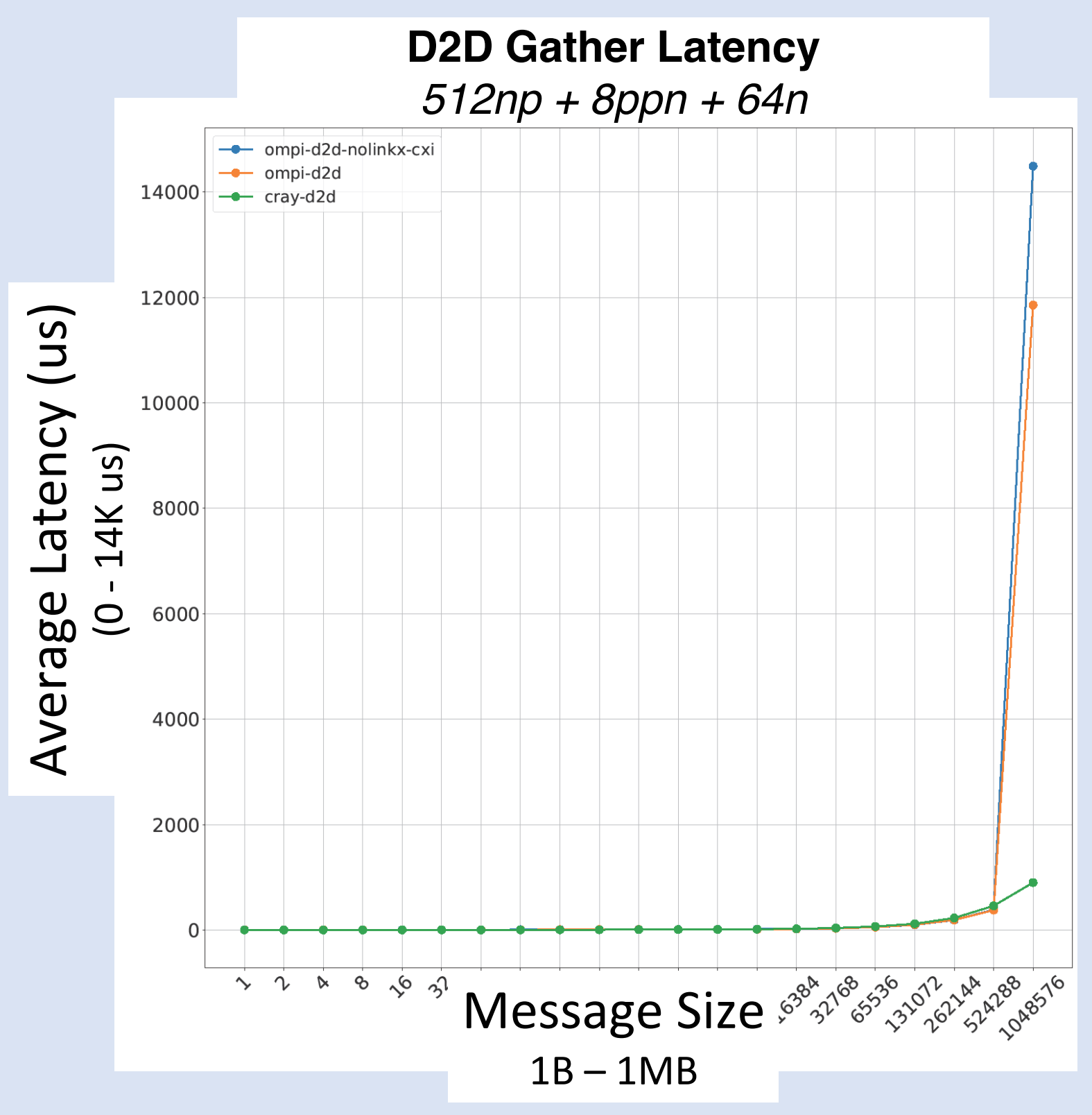
Completed

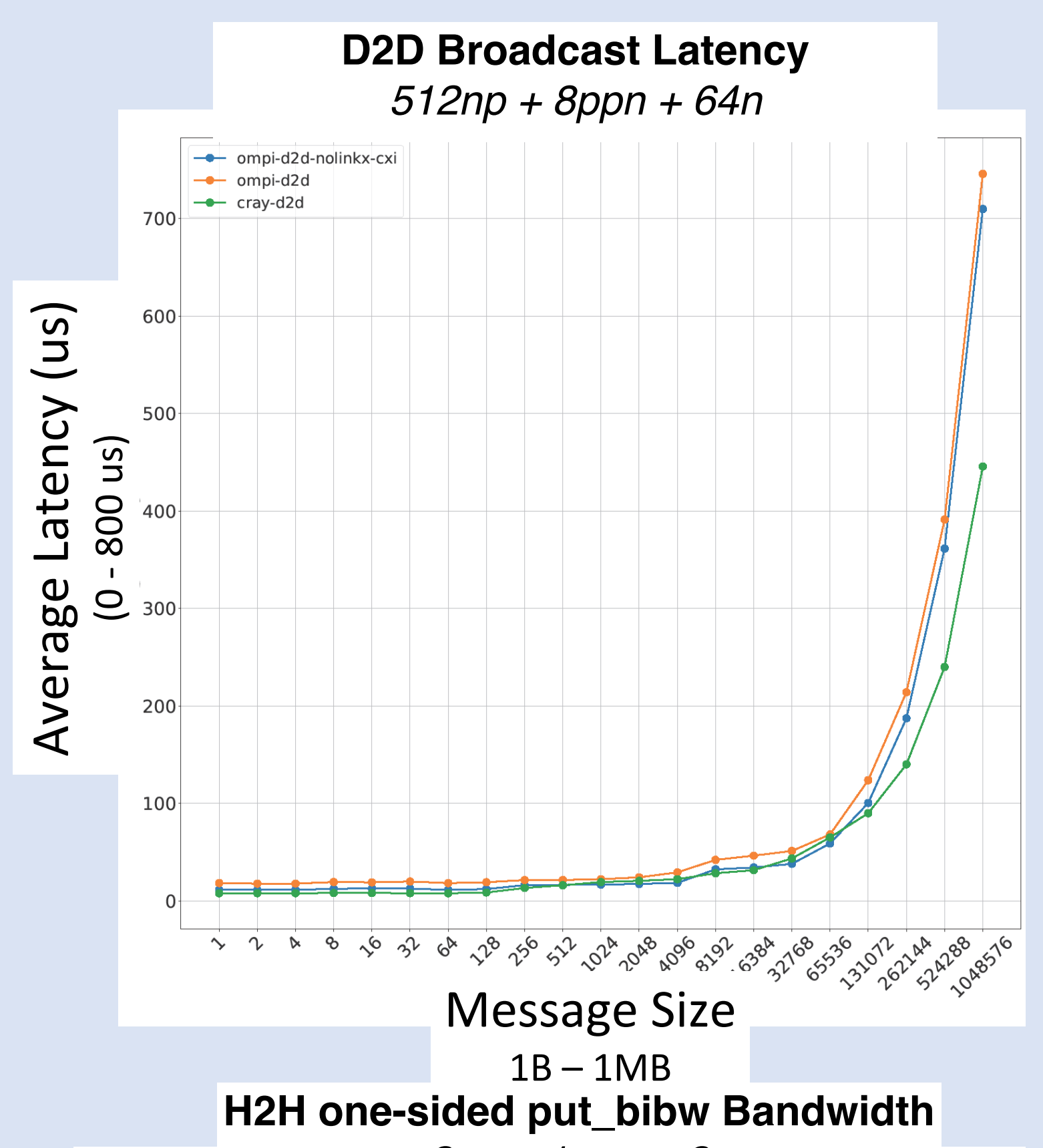
- Shared completion & receive queues
- SHM features (ROCM support, Asynchronous IPC, etc.)
- SHM Locking improvements
- LINKx with tagged message support
- LINKx with RMA support
- OMPI Process/NIC binding

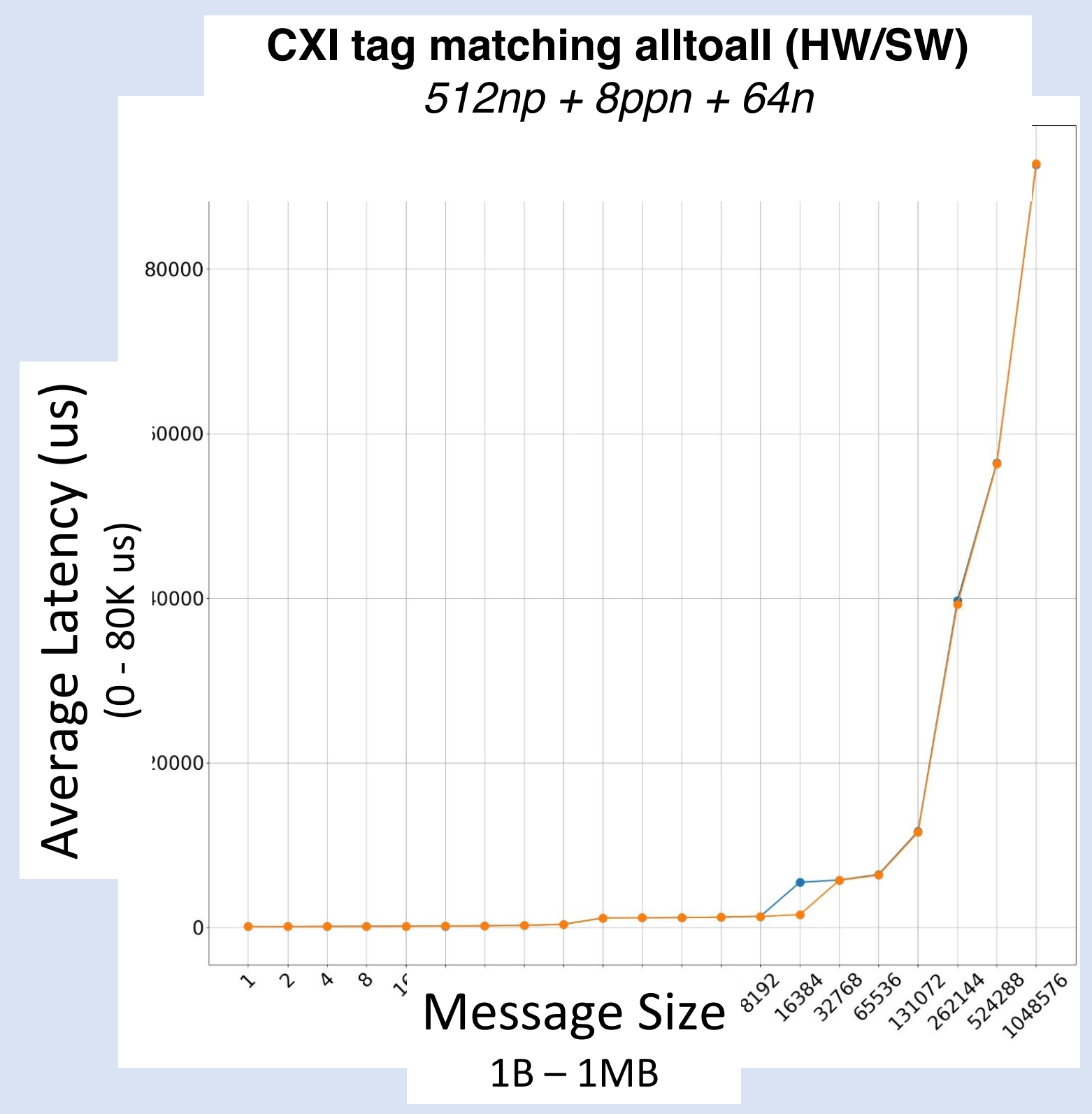
Future Work

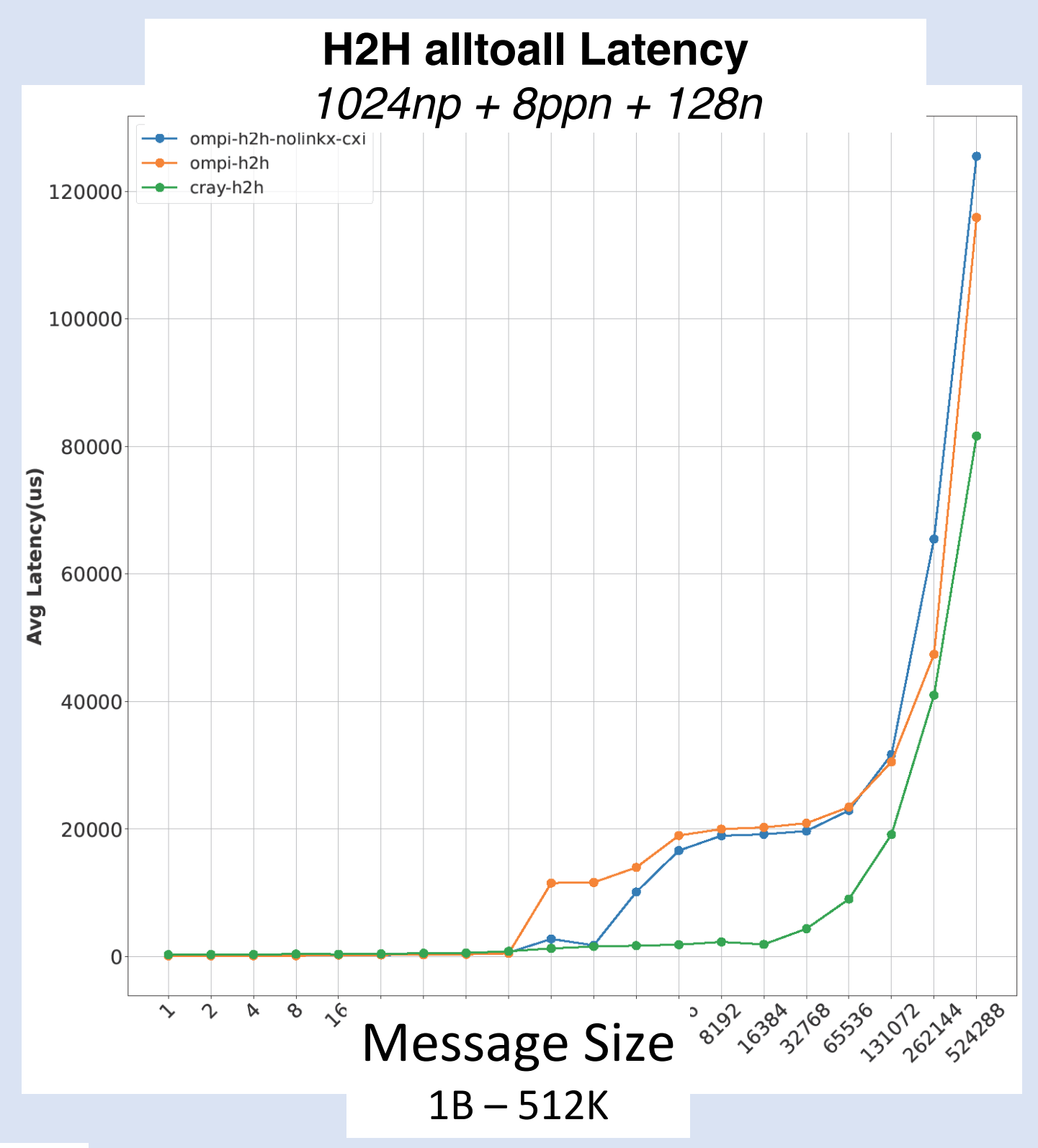
- Refining provider selection at the OMPI layer
- Support Intel GPUs on Aurora
- Add libfabric Peer APIs for Memory Region Registration
- Performance tuning (eg: Small Messages, Collectives)

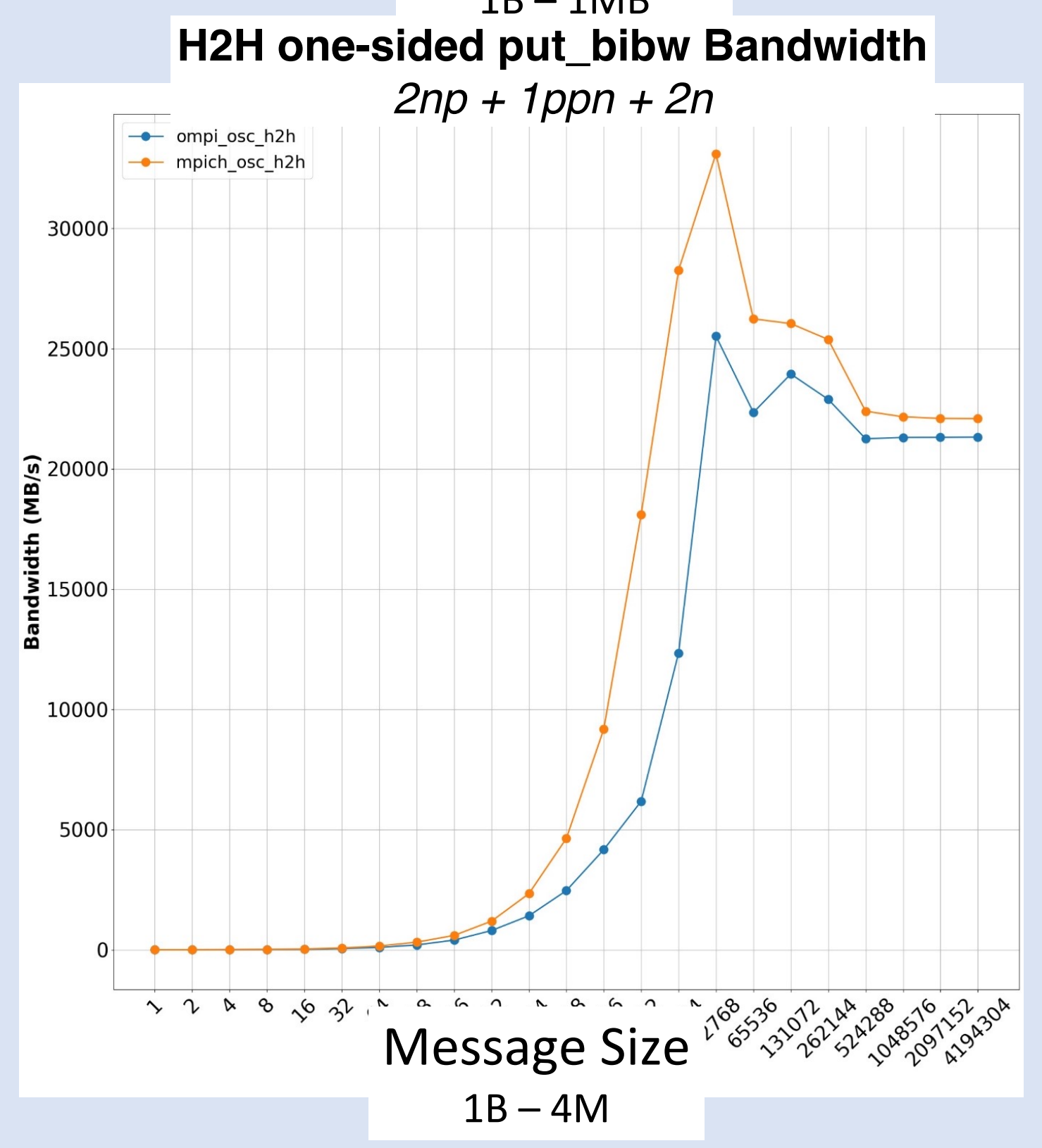












This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.