# Python Micro-Benchmarks for Evaluating MPI Libraries on HPC Systems

Talk at OSU Booth SC '22

by Nawras Alnaasan

Network-Based Computing Laboratory

Dept. of Computer Science and Engineering , The Ohio State University

alnaasan.1@osu.edu

# OSU MPI Micro-Benchmarks (OMB) Suite

- OMB is a benchmarking tool that aids in measuring the performance of communication libraries on HPC systems with different configurations and hardware
- There is support for a variety of programming models and communication libraries including MPI, OpenSHMEM, UPC, and UPC++
- It provides a variety of benchmarks including point-to-point, blocking/non-blocking collectives, and one-sided communication primitives
- There is support for evaluation performance of data communication to/from NVIDIA and AMD GPUs
- The aim of this talk is to provide an overview of recent Java and Python extensions to OMB

# Python and Java Extensions to OMB

- Java and Python extensions have been released as part of the OMB 6.0 release:
  - https://mvapich.cse.ohio-state.edu/benchmarks/
- Instructions for using OMB for Java:
  - User guide: https://mvapich.cse.ohio-state.edu/static/media/mvapich/README-OMB-J.txt
  - Sample run:

```
mpirun_rsh -np 2 -hostfile hosts \
  LD_PRELOAD=${MPILIB}/lib/libmpi.so java -cp $MV2J_HOME/lib/mvapich2-j.jar:. \
  -Djava.library.path=$MV2J_HOME/lib mpi.pt2pt.OSUBandwidth
```

- Instructions for using OMB for Python:
  - User guide: https://mvapich.cse.ohio-state.edu/static/media/mvapich/README-OMB-PY.txt
  - Sample run:

```
mpirun -np 2 --hostfile hosts python run.py \
--benchmark latency --buffer numpy
```

# High Performance Computing with Python

- Python has become a dominant programming language for emerging areas like Machine Learning (ML), Deep Learning (DL), and Data Science (DS).
- Python has a rapidly growing community and support for prominent scientific libraries and frameworks with a flexible and simplified syntax.
- ML, DL, and DS applications are computationally intensive tasks that can be accelerated by harnessing the compute power offered by HPC.


Courtesy: https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c

# Why Python?

Flexible and simplified syntax.

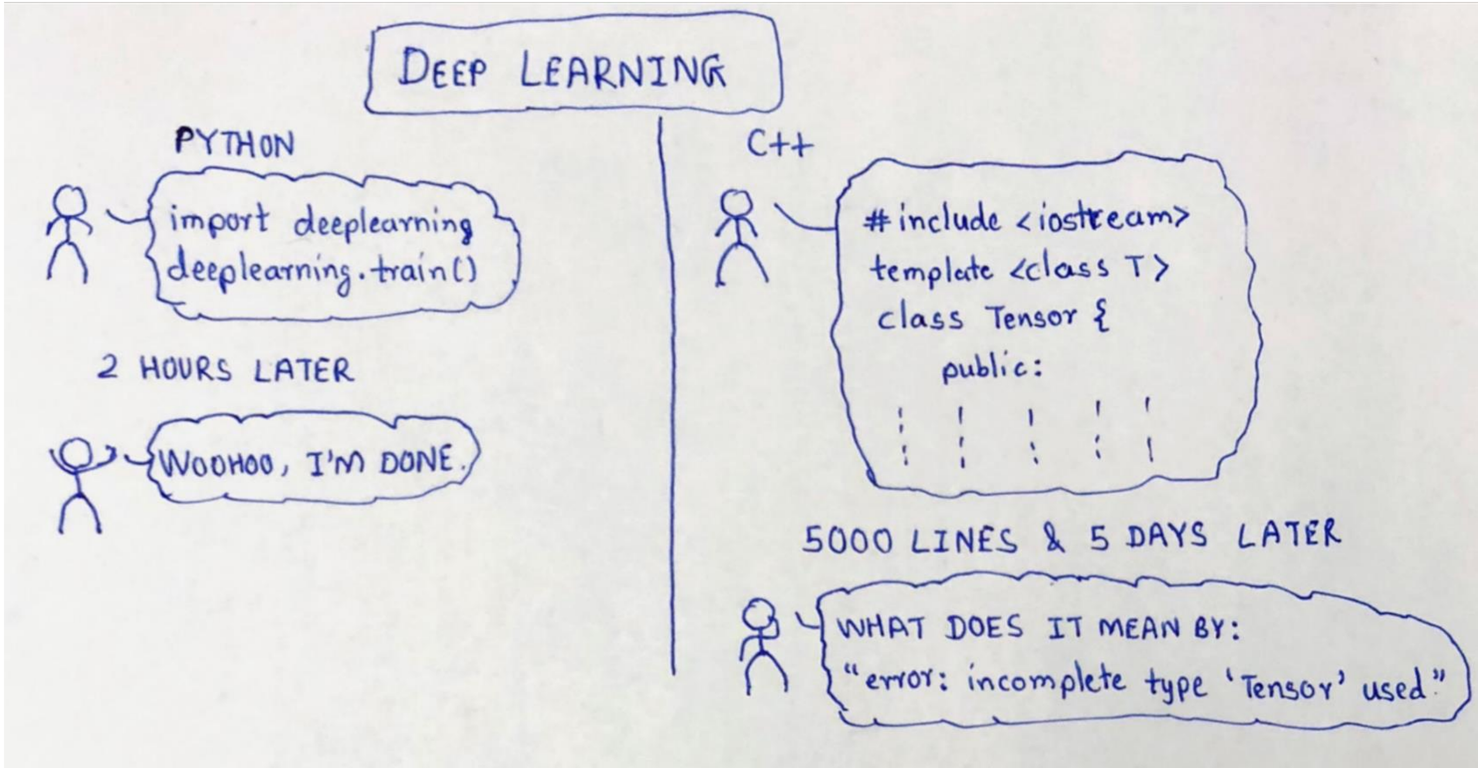
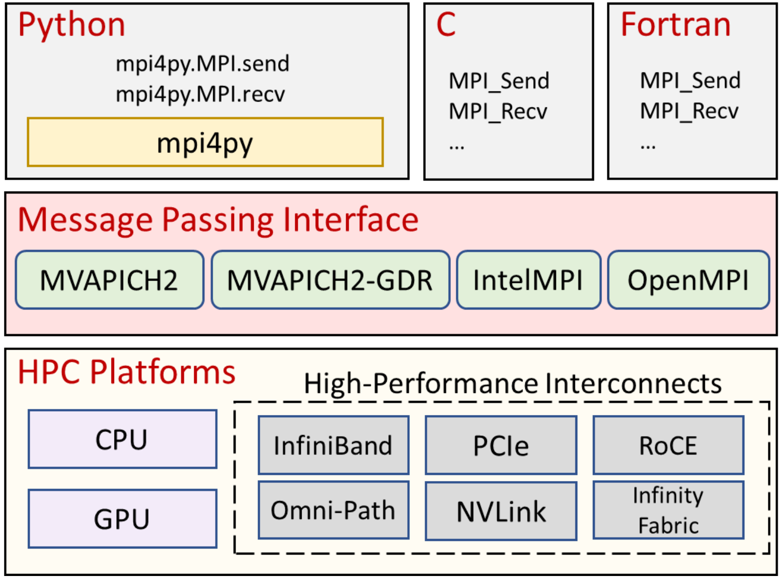



Image source: https://www.hardikp.com/2017/12/30/python-cpp/

# MPI for Python

- Message Passing Interface (MPI) is considered the de-facto standard that defines communication operations for exchanging data in parallel computing environments.
- The MPI standard only provides bindings for the C and Fortran programming languages.
- To use MPI with higher-level programming languages such as Python, a communication wrapper library is needed to provide MPI-like bindings.
- mpi4py is a widely used package that provides a Python-based MPI interface which is built on top of an MPI library.




C and Fortran can directly call MPI operations whereas Python needs a wrapper to provide MPI-like bindings.

# Package Comparison

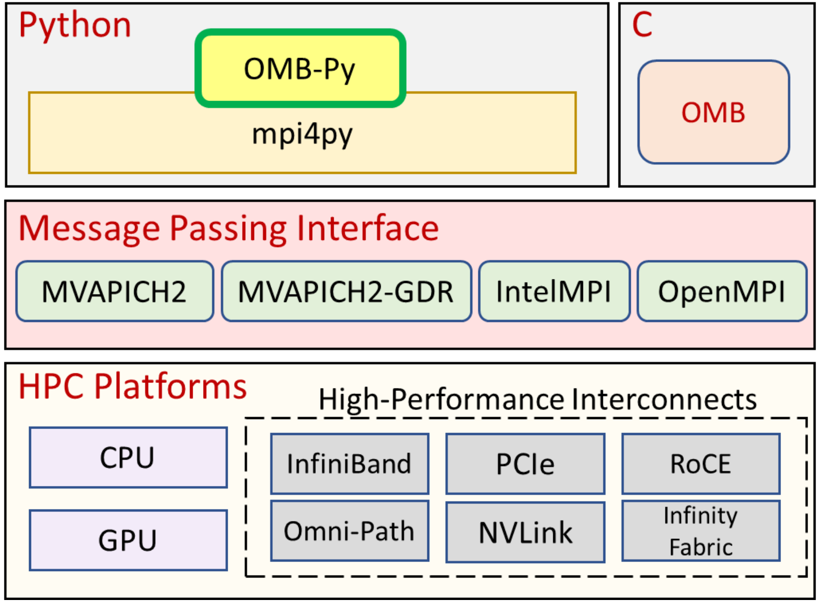
- Design and implementation of OMB-Py—a Python extensions to the open-source OMB suite—aimed to evaluate communication performance of MPI-based parallel applications in Python.
- Performance characterization of MPI communication in Python on four HPC systems:
  - Point-to-point and collective communication operations using OMB as a baseline performance in C.
  - Evaluation on CPU and GPU devices for different buffers including Bytearrays, Numpy, CuPy, PyCUDA and Numba.
  - Pickle method evaluation for serializing communicated objects.
- Analysis of the overhead presented by mpi4py over native MPI libraries.

| | Point-to-point | Blocking Collectives | Vector Variants | Support for Python | Bytearray Buffers | Numpy Buffers | CuPy Buffers | PyCUDA Buffers |
|---|---|---|---|---|---|---|---|---|
| OMB-Py (Proposed Design) | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| mpi4py Demo Codes [1] | ✓ | Partial | Partial | ✓ | X | ✓ | X | X |
| IMB [2] | ✓ | ✓ | ✓ | X | X | X | X | X |
| SMB [3] | ✓ | X | X | X | X | X | X | X |

**Feature Comparison Between Benchmark Packages**

[1] L. Dalcin, R. Paz, and M. Storti, MPI for Python, Journal of Parallel and Distributed Computing, 65(9):1108-1115, 2005. https://doi.org/10.1016/j.jpdc.2005.03.010
[2] "Sandia MPI Micro-Benchmark Suite (SMB)." http://www.cs.sandia.gov/smb/index.html
[3] "Intel MPI Benchmarks (IMB)." https://software.intel.com/en-us/articles/intel-mpi-benchmarks

# Hierarchy and Supported Benchmarks




**Architectural hierarchy of OMB-Py with mpi4py, MPI, and HPC platforms**

| | |
|---|---|
| Point-to-Point | Bi-directional bandwidth, Bandwidth, Latency, Multi latency |
| Blocking Collectives | Allgather, Allreduce, Alltoall, Barrier, Bcast, Gather, Reduce_scatter, scatter |
| Vector Variant | Allgatherv, Alltoallv, Gatherv, Scatterv |

**Point-to-Point, blocking collectives, and vector variant benchmarks supported by the OMB-Py package**

N. Alnaasan , A. Jain , A. Shafi , H. Subramoni , and DK Panda, OMB-Py: Python Micro-Benchmarks for Evaluating Performance of MPI Libraries on HPC Systems

# Benchmarking and Sample Run

- Maintained similar approach to OMB but in Python for fair comparison.
- MPI_Barrier() guarantees that both sender and receiver processes start at the same time.
- Latency is averaged across multiple iterations.
- MPI_Reduce() is used to aggregate averages across all participating processes.

```
init_MPI_communication(...);
allocate(s_buf, ...);
allocate(r_buf, ...);
for size in message_sizes do
    MPI_Barrier();
    if myrank == 0 then
        start_time = current_time();
        for i: 1 ... max_iterations do
            MPI_Send(s_buf, size ...);
            MPI_Recv(r_buf, size ...);
        end
        end_time = current_time();
        latency = (start_time - end_time);
    else
        start_time = current_time();
        for i: 1 ... max_iterations do
            MPI_Recv(r_buf, size ...);
            MPI_Send(s_buf, size ...);
        end
        end_time = current_time();
        latency = (start_time - end_time);
    end
    latency = latency / (2 * max_iterations);
    report_latency();
end
```

**Algorithm for Blocking Send/Recv Latency Benchmark**

```
# OMB-Py MPI Latency Test
# Size [B]          Latency [us]
0                          1.55
1                          1.58
2                          1.58
4                          1.58
8                          1.58
16                         1.62
32                         1.62
64                         1.63
128                        1.68
256                        2.10
512                        2.18
1024                       2.36
2048                       2.73
4096                       3.63
8192                       5.03
16384                      7.24
32768                      9.79
65536                     12.73
131072                    22.17
262144                    33.01
524288                    55.19
1048576                   97.53
2097152                  186.07
4194304                  354.46
```

Users can customize their runs by using runtime flags:

- Device: either CPU or GPU device on each node.
- Buffer: can choose from a list of Python objects including Numpy, CuPy, PyCUDA, Numba, etc.
- Message size: define lower and upper limits for message sizes to report.
- Number of iterations: number of times the tested operation is executed.

**Sample output of point-to-point blocking latency test**

N. Alnaasan , A. Jain , A. Shafi , H. Subramoni , and DK Panda, OMB-Py: Python Micro-Benchmarks for Evaluating Performance of MPI Libraries on HPC Systems

# Experimental Setup

| | Frontera | Stampede2 | RI2 | Bridges-2 |
|---|---|---|---|---|
| CPU | Two Intel Xeon Platinum 8280 (Cascade Lake). 28 cores per socket (56 per node) @2.70GHz | Two Intel(R) Xeon(R) Platinum 8160 (Skylake). 24 cores per socket (48 per noe) @2.70GHz | Two Intel(R) Xeon(R) Gold 6132 with 14 cores (28 cores per node) @2.40GHz. | Two Intel Xeon Gold 6248 (Cascade Lake). 20 cores per socket (40 cores per node) @ 2.50GHz |
| RAM | 192GB of RAM per node. | 192GB of RAM per node. | 128GB of RAM per node | 512GB of RAM per node. |
| Interconnect | Mellanox InfiniBand HDR | Intel Omni-Path | Mellanox InfiniBand | Mellanox InfiniBand HDR |
| GPU | N/A | N/A | N/A | Eight NVIDIA Tesla V100-32GB SXM2 per node |

**Configuration of nodes used on each of the experimental systems**

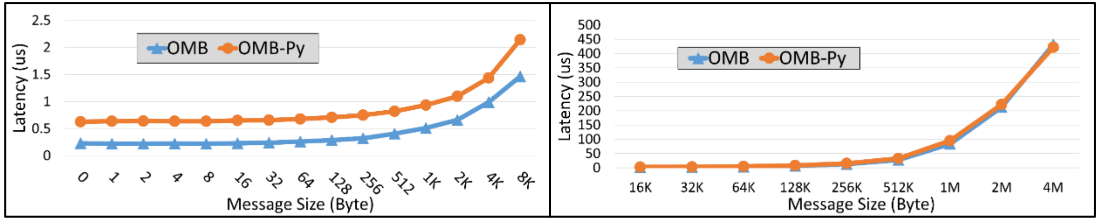
Software packages:

- For CPU experiments: MVAPICH2 2.3.6, OMB v5.8, mpi4py 3.1.1
- For GPU experiments: MVPICH2-GDR 2.3.6, CUDA 11.2, OMB v5.8, mpi4py 3.1.1

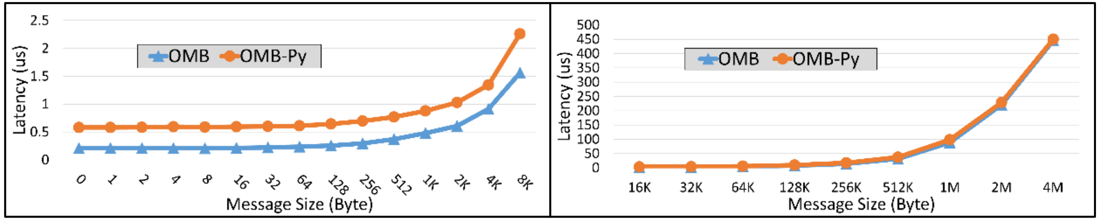
N. Alnaasan , A. Jain , A. Shafi , H. Subramoni , and DK Panda, OMB-Py: Python Micro-Benchmarks for Evaluating Performance of MPI Libraries on HPC Systems

# Point-to-Point Evaluation on CPU

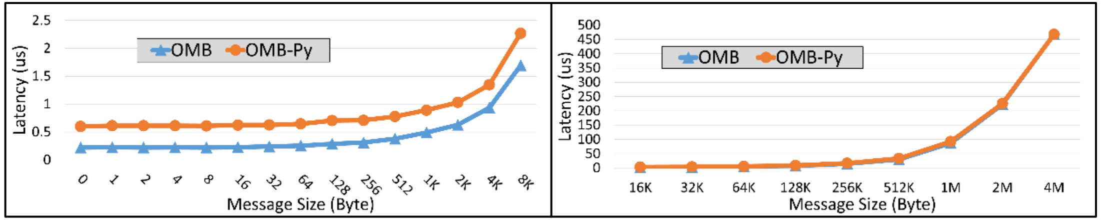
CPU communication latency for small and large message sizes comparing OMB and OMB-Py benchmarks




**Frontera**



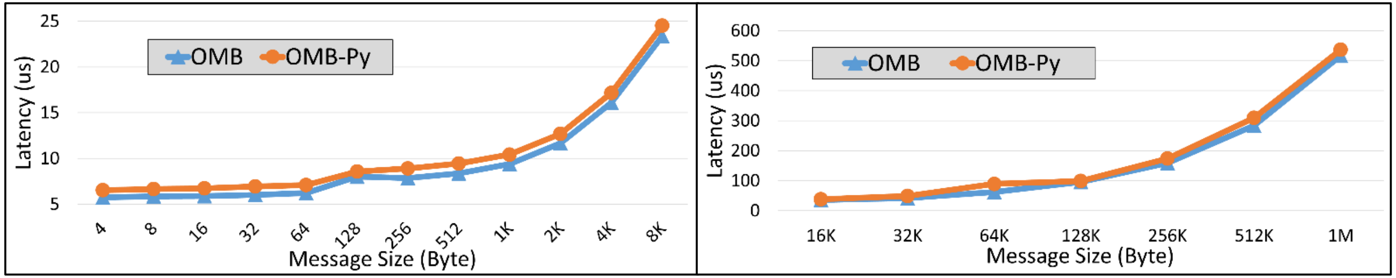

**Stampede2**



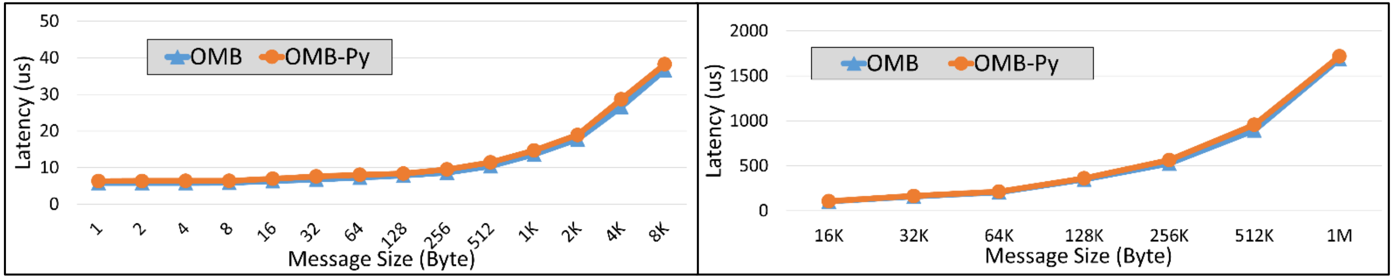

**RI2**

- Consistent trends across three clusters.
- Average overhead of 0.44, 0.41, and 0.41 on Frontera, Stampede2, and RI2 respectively for small message sizes and 2.31, 4.13, 1.76 microseconds for large message sizes.

# Collectives Evaluation on CPU




Allreduce CPU communication latency for small and large message sizes on 16 nodes on the Frontera Cluster

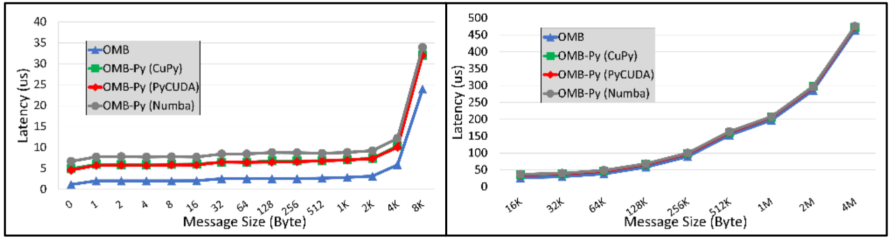



Allgather CPU communication latency for small and large message sizes on 16 nodes on the Frontera Cluster

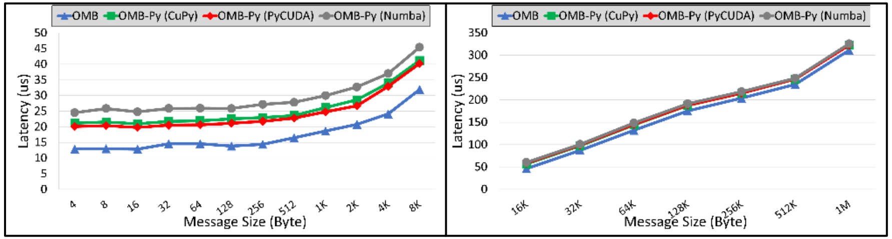
Average overheads of 0.93 and 0.92 microseconds for Allreduce and Allgather respectively for small message sizes. 14.13 and 23.4 for large message sizes.

N. Alnaasan , A. Jain , A. Shafi , H. Subramoni , and DK Panda, OMB-Py: Python Micro-Benchmarks for Evaluating Performance of MPI Libraries on HPC Systems

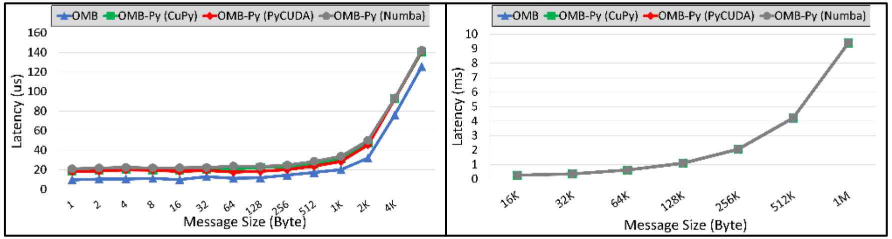
# Evaluation on GPU




Point-to-Point GPU communication latency for small and large message sizes on Bridges-2




Allreduce GPU communication latency for small and large message sizes on Bridges-2 (2 nodes – 8 GPUs)



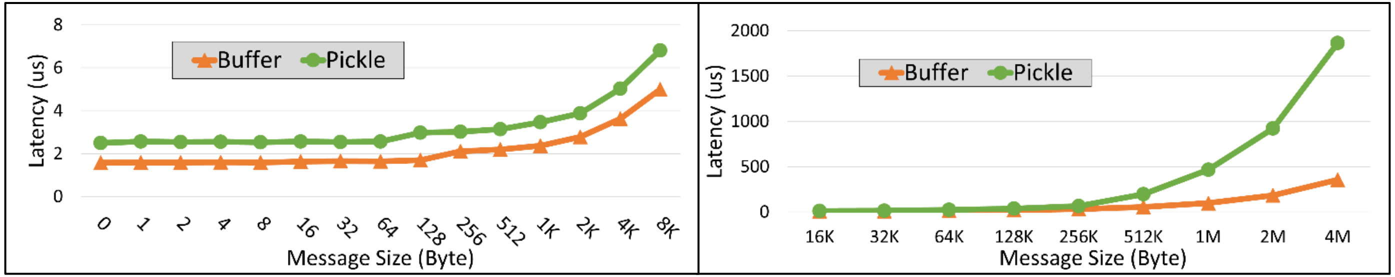

Allgather GPU communication latency for small and large message sizes on Bridges-2 (2 nodes – 8 GPUs)

CuPy, PyCUDA, and Numba libraries allow initializing different types of data buffers directly on the GPU to carry out complex matrix operations. In these benchmarks, communication happens directly from/to the GPU by utilizing these GPU buffers.

Across all benchmarks, CuPy and PyCUDA show better MPI communication performance on the GPU compared to Numba.

# Pickle Method Evaluation

- mpi4py offers a built-in feature for serialization of the communicated Python objects.
- This is mainly referred to as "pickling" when an object is converted into a byte stream and "unpickling" when it is converted back to its original format.
- In mpi4py, the MPI methods that use the pickle method are defined with a lower case first letter such as send(), recv(), reduce(), allgather, etc. The direct buffer methods (no serialization) are defined with upper case first letter such as Send(), Recv(), Reduce(), Allgather(), etc.

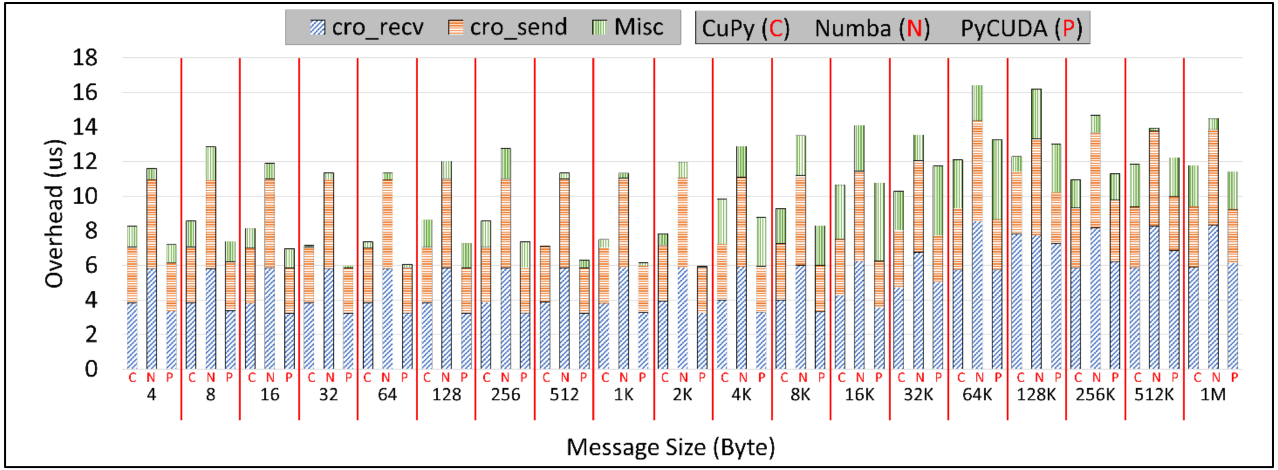



**CPU latency for small and large message sizes using OMB-Py to compare the pickle and direct buffer methods on Frontera**

N. Alnaasan , A. Jain , A. Shafi , H. Subramoni , and DK Panda, OMB-Py: Python Micro-Benchmarks for Evaluating Performance of MPI Libraries on HPC Systems

# Overhead Analysis

- In order to determine the source of overhead caused by the Python/Cython layer over the native MPI libraries, we perform comprehensive profiling of the mpi4py Allreduce function for the CuPy, Numba, and PyCUDA buffers.
- The Allreduce function in mpi4py consists of two phases: 1) a staging phase to perform checks and links of the Python send and receive buffers in Cython, 2) an execution phase which mainly calls the implementation of the MP operation provided by the underlying MPI library.
- The following analysis shows that 80% to 90% of the overall overhead is spent on preparing the send and receive buffers.




Allreduce GPU overhead analysis using CuPy, Numba, and PyCUDA buffers on 16 GPUs (2 nodes - 8 GPUs per node) on the Bridges-2 cluster

# Thank You!

alnaasan.1@osu.edu

Network-Based Computing Laboratory
http://nowlab.cse.ohio-state.edu/

High Performance Deep Learning
http://hidl.cse.ohio-state.edu/

The High-Performance Deep Learning Project
http://hidl.cse.ohio-state.edu/

The High-Performance MPI/PGAS Project
http://mvapich.cse.ohio-state.edu/