Characterizing One-/Two-sided Designs in OpenSHMEM Collectives

Yuke Li
University of California, Merced
Merced, California, USA
yli304@ucmerced.edu

Yanfei Guo
Argonne National Laboratory
Lemont, IL, USA
yguo@anl.gov

Xiaoyi Lu
University of California, Merced
Merced, California, USA
xiaoyi.lu@ucmerced.edu

ABSTRACT

OpenSHMEM is a widely used Partitioned Global Address Space (PGAS) programming model in the HPC community. The latest OpenSHMEM Specification v1.5 introduced the team concept and team-based collective communication that are similar to the communicator and collective communication in the Message Passing Interface (MPI) programming model. However, the typical design of OpenSHMEM collectives relies on one-sided communication such as `Put` and `Get` to move the data, which is different from two-sided communication in MPI collectives. In this work, we compare OpenSHMEM collective designs using native one-sided communication and MPI-based two-sided communication on an HPC cluster. We characterize two aspects (i.e., synchronization and collective algorithms) that can influence the performance of these two different designs and use benchmarks to show the performance differences. Through our evaluation, we find that the MPI-based design is faster than the one-sided design at most times, while the one-sided design can perform faster in certain cases.

1 INTRODUCTION

Partitioned Global Address Space (PGAS) [2] is a popular parallel programming model for High Performance Computing (HPC) applications. OpenSHMEM is a widely used specification of the PGAS model. It implements PGAS by defining remotely accessible memory buffers to share information among OpenSHMEM processes, or Processing Elements (PEs) [1]. The latest OpenSHMEM Specification version 1.5 introduces the *team* concept and team-based collective communication, which uses teams to group OpenSHMEM PEs and identify which PEs should participate in the collective communication.

Typically, the OpenSHMEM collectives are implemented with one-sided communication. For example, a broadcast operation can be implemented as the root process performing a series of `Puts` in the shared global memory of other processes as shown in Figure 1(a). In contrast, MPI collectives are implemented using two-sided communication where a source PE calls a *send* that will be matched with a *recv* called by the target PE as shown in Figure 1(b). These two different design approaches can lead to different performance characteristics of collectives on HPC clusters (even for the same collective routine) due to their varied communication algorithms and synchronization costs. Therefore, comparing these two different designs and understanding their performance characteristics are essential for further research and development of OpenSHMEM.

In this work, we aim to answer two important questions: *(1) How big it is for the performance difference between these two designs? (2) What are the potential reasons for these performance gaps?* Hence, we first compare OpenSHMEM collective designs using

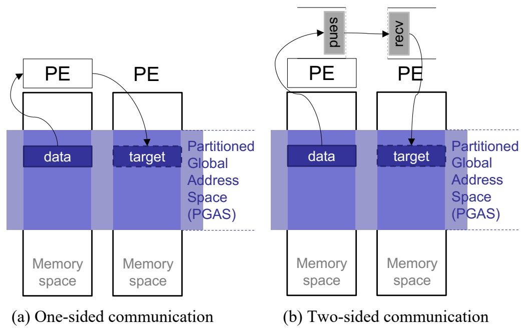

PE
PE
data
target
Partitioned Global Address Space (PGAS)
Memory space
Memory space
(a) One-sided communication
send
recv
PE
PE
data
target
Partitioned Global Address Space (PGAS)
Memory space
Memory space
(b) Two-sided communication

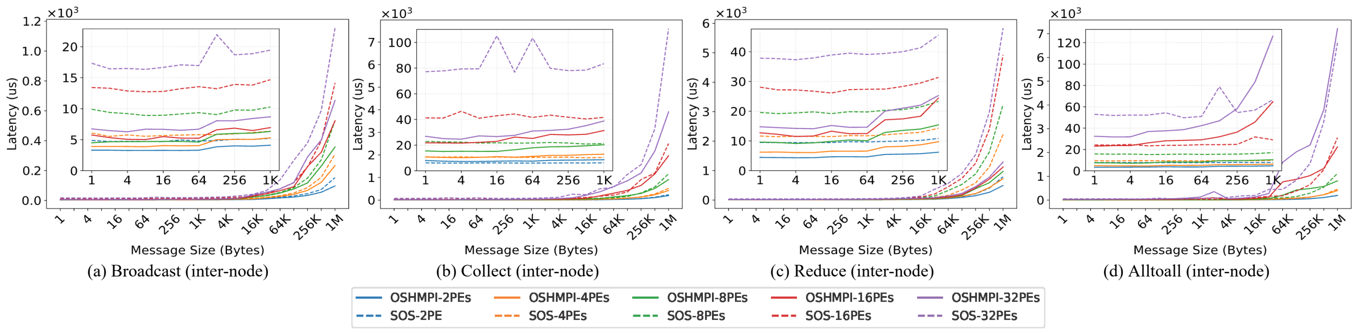
Figure 1: Two different designs of OpenSHMEM collectives.

native one-sided communication (i.e., the Sandia OpenSHMEM (SOS) library [4]) and MPI-based two-sided communication (i.e., the OSHMPI library [7, 10]) on a modern HPC cluster. The OSHMPI library is an OpenSHMEM implementation on top of MPI which allows us to focus on analyzing the performance difference due to the implementation of collective operations.

2 DESIGNS OF OPENSHMEM LIBRARIES

In the OpenSHMEM design, which is based on one-sided communication, `Put` and `Get` operations are employed. To complete the data movements and synchronize all Processing Elements (PEs), `SHMEM_WAIT_UNTIL` is used subsequently. The synchronization in one-sided design does not require the target PE to actively acknowledge the data received from source PEs. This flexibility in one-sided synchronization could allow PEs to overlap the data movement among different targets more efficiently, especially when the collective communication is not well-balanced. This is significantly different from the two-sided communication based collective design because its data movement happens in multiple coordinated stages of *send/recv* among all of the processes.

Another important performance aspect is the collective algorithm. Generally, most OpenSHMEM collectives select the algorithms such as tree, ring, or linear (round-robin). OSHMPI collectives essentially call MPI collectives. For example, `shmem_alltoall` calls `PMPI_Ialltoall` internally. Taking MPICH as an example, it provides rich collective algorithms for ring, Brucks [3], recursive doubling, etc [11]. The algorithm can be chosen at runtime based on scheduling factors, like how many PEs are participating and how large the message size is. Different algorithm selections could result in different performance results. For example, in a two-sided communication, OSHMPI `alltoall` might choose a tree model that needs multiple stages to complete because of the PEs organization in a tree and there are multiple *sends* and *recvs* between the



(a) Broadcast (inter-node) (b) Collect (inter-node) (c) Reduce (inter-node) (d) Alltoall (inter-node)

Figure 2: The collective communication performance comparison with OSHMPI and SOS up to 32 nodes (inter-node).

Table 1: Summary of key findings for two types of OpenSHMEM designs.

Key Findings	Reason Explanations
OSHMPI (two-sided) is faster than SOS (one-sided) in most cases.	The existing two-sided design (e.g., OSHMPI) can inherit the advantages in well-optimized MPI engine, while the one-sided design (e.g., SOS) still needs optimized implementations.
SOS shows lower or comparable latency in certain cases: - Communication with a small number of PEs like 2, for *collect*, intra-node *broadcast*, *alltoall*, and *reduce* collectives. - The *alltoall* collective for medium messages in intra-node communication and medium-large messages in inter-node communication with more PEs, like 16 and 32.	With a small number of PEs, basic point-to-point primitive performance and synchronization method dominate the performance. SOS's native one-sided design can achieve better overlapping with simpler synchronizations. Besides, SOS also has some specific optimizations on Omni-Path Fabric. For the *alltoall* collective, MPI internal routine needs further optimization or tuning for these particular settings.
SOS shows unstable performance in the case of intra-node communication with 32 PEs.	SOS uses extra helper threads to progress the internal communication tasks and thus it is oversubscribed.

stages. However, the one-sided SOS design could simply use the round-robin algorithm and put the data to each PE's destination buffer.

3 PERFORMANCE CHARACTERIZATION

We characterize the performance of OSHMPI and SOS on the Bebop [8] HPC cluster with up to 32 nodes. Each node is equipped with 36-core Intel Broadwell Xeon E5-2695v4 CPUs and 128GB DDR4 DRAM. The nodes are connected with Omni-Path Fabric. We use MPICH [5, 6] as the MPI engine. We chose OSU Micro-Benchmarks (OMB) [9] to run the experiments. The OpenSHMEM collective benchmarks in OMB do not support the team-based collectives in OpenSHMEM specification v1.5 yet. Hence, we modify the code in OMB to match the requirements in the new specification.

We run the evaluation for both intra-node communication and inter-node communication. In addition, we set the corresponding environment variable to make sure all the traffic goes through the interconnect instead of local copy through the shared memory. The collective communication performance characteristics of OSHMPI and SOS on Bebop are illustrated in Figure 2. Due to the page limit, we only include the inter-node performance results in this paper as shown in Figure2. The intra-node performance results are shown in the poster.

In a nutshell, the most obvious characteristic shown in Figure2 is that OSHMPI (two-sided design) outperforms SOS (one-sided design) in most cases, while SOS is faster or comparable in certain cases. One of the biggest performance gaps happens in the case of intra-node *collect* with 32PEs, where OSHMPI achieves 1/10 latency of SOS. In the case of inter-node *alltoall* with more than 16 PEs, SOS achieves 1/4 latency of OSHMPI. We summarize our key findings and the corresponding reason explanations in Table 1.

4 CONCLUSION

In this paper, we find that the performance of state-of-the-art OpenSHMEM libraries that are designed in two different ways (one-sided and two-sided) varies a lot (up to 10X) with several important factors, such as the number of PEs, the size of messages, different synchronization methods, and different collective algorithms. We use collective benchmarks to compare the latency of different OpenSHMEM designs and expose their performance characteristics. We believe the conducted extensive performance characterizations in this paper can give the community some insights for future research avenues on OpenSHMEM designs and optimizations.

ACKNOWLEDGMENTS

This research was 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, and by the U.S. Department of Energy, Office of Science, under Contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided on Bebop, a HPC cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory. This work was supported in part by an NSF research grant OAC #2321123 and a DOE research grant DE-SC0024207.

REFERENCES

[1] 2020. *OpenSHMEM Application Programming Interface, v1. 5 Final.* Technical Report.

[2] George Almasi. 2011. *PGAS (Partitioned Global Address Space) Languages.* Springer US, Boston, MA, 1539–1545. https://doi.org/10.1007/978-0-387-09766-4_210

[3] J. Bruck, Ching-Tien Ho, S. Kipnis, E. Upfal, and D. Weathersby. 1997. Efficient Algorithms for All-To-All Communications in Multiport Message-Passing Systems. *IEEE Transactions on Parallel and Distributed Systems* 8, 11 (1997), 1143–1156. https://doi.org/10.1109/71.642949

[4] Sandia Corporation. 2023. Sandia-OpenSHMEM/SOS. https://github.com/Sandia-OpenSHMEM/SOS

[8] Argonne National Laboratory. 2023. Bebop. https://www.lcrc.anl.gov/systems/resources/bebop/

[5] Argonne National Laboratory. 2023. MPICH: High-Performance Portable MPI. https://www.mpich.org/

[6] Argonne National Laboratory. 2023. Official MPICH Repository. https://github.com/pmodels/mpich

[7] Argonne National Laboratory. 2023. OSHMPI: OpenSHMEM Implementation over MPI. https://github.com/pmodels/oshmpi

[9] Dhabaleswar Kumar Panda, Hari Subramoni, Ching-Hsiang Chu, and Mohammadreza Bayatpour. 2021. The MVAPICH project: Transforming research into high-performance MPI library for HPC community. *Journal of Computational Science* 52 (2021), 101208. https://doi.org/10.1016/j.jocs.2020.101208 Case Studies in Translational Computer Science.

[10] Min Si, Huansong Fu, Jeff R. Hammond, and Pavan Balaji. 2022. OpenSHMEM over MPI as a Performance Contender: Thorough Analysis and Optimizations. In *OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Exascale and Smart Networks*, Stephen Poole, Oscar Hernandez, Matthew Baker, and Tony Curtis (Eds.). Springer International Publishing, Cham, 39–60.

[11] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *Int. J. High Perform. Comput. Appl.* 19, 1 (feb 2005), 49–66. https://doi.org/10.1177/1094342005051521