shmem4py: OpenSHMEM for Python

Marcin Rogowski [1*¶], **Lisandro Dalcin** [1*], **Jeff R. Hammond** [2], and **David E. Keyes** [1]

1 King Abdullah University of Science and Technology, Saudi Arabia **2** NVIDIA Helsinki Oy, Finland ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.05444

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @greghbauer
- @gonsie

Submitted: 02 May 2023
Published: 19 July 2023

Summary

shmem4py brings the Partitioned Global Address Space (PGAS) programming model to Python by exposing the functionality of the OpenSHMEM Application Programming Interface (API) specification. The feature set includes one-sided communication, shared memory access, atomic memory operations, and collective operations. The Python implementation of shmem4py emphasizes using NumPy arrays, providing convenient access to the symmetric memory allocations central to OpenSHMEM's programming model. Thanks to Python's versatility and OpenSHMEM implementations' focus on performance, shmem4py offers a seamless experience on a variety of hardware, from laptops to supercomputers, and for a wide range of applications and users. shmem4py API grounds in OpenSHMEM 1.5 specification; however, it also supports legacy 1.4 implementations.

Statement of Need

Python applications can be scaled to multiple processes and compute nodes in various ways. When working on a single node, the `multiprocessing` or `concurrent.futures` packages from the Python standard library offer solutions for task-based parallelism. As we expand beyond a single node, more advanced frameworks like Dask (Rocklin, 2015), Ray (Moritz et al., 2018) or `mpi4py.futures` (Rogowski et al., 2023) are commonly used. Typically, these high-level frameworks handle interprocess communication transparently to the user.

More challenging applications often require specialized communication patterns. In those cases, Python applications can leverage communication frameworks originally designed for high-performance computing. One such example is `mpi4py` (Dalcin & Fang, 2021), which offers MPI bindings for Python. `shmem4py` adopts a similar approach, providing Python bindings to OpenSHMEM (Chapman et al., 2010) implementations with a Python-centric and high-level API built on top of a low-level CFFI (Rigo & Fijalkowski, 2022) module. This way, `shmem4py` is accessible to a diverse audience while offering a proven programming model with reliable performance.

`shmem4py` complements `mpi4py` in the same way that OpenSHMEM complements MPI. MPI is extremely popular in high-performance computing because of its combination of a rich feature set and generality. OpenSHMEM is based on a different philosophy, which provides a smaller set of features that better match high-performance computing system hardware capabilities. For example, OpenSHMEM one-sided communication and atomic operations are aligned with networking capabilities such as remote direct memory access (RDMA) such that no intervention is required on the remote side (Jose et al., 2014). In contrast, MPI's one-sided communication functionality includes many more features, some of which are known to make implementations based strictly on RDMA more difficult (Hammond et al., 2014; Si et al., 2015). Another aspect of the tradeoff between generality and close-to-hardware features is observed in the context of specialized processors, such as GPUs. It has been shown that OpenSHMEM-like APIs can be implemented natively on GPUs, e.g., NVSHMEM (Hsu et al., 2020; Potluri et al., 2017). On the other hand, implementing MPI send and receive operations in the same context poses significant challenges. In fact, `shmem4py` will provide the backbone for future extensions supporting inter-GPU communication.

We envision two distinct groups of users that may be interested in `shmem4py`. The first, and likely most numerous, group includes Python programmers who lack the expertise or the time to write low-level code in C and have applications well-suited for the PGAS paradigm. The second group comprises high-performance computing professionals familiar with OpenSHMEM who want to prototype or port parts of their applications to Python. Both groups of users can benefit greatly from the ease of development in Python, `shmem4py`'s convenience functions for manipulating NumPy arrays in symmetric memory, and all of OpenSHMEM features. We also expect that current users of `mpi4py` may want to try `shmem4py` as a complementary communication model for the same reasons that OpenSHMEM is used alongside MPI.

Supported OpenSHMEM Implementations

The `shmem4py` package supports, and is tested with, all major implementations of the OpenSHMEM specification:

- Cray OpenSHMEMX
- Open MPI OpenSHMEM
- Open Source Software Solutions (OSSS) OpenSHMEM
- OSHMPI
- Sandia OpenSHMEM

Acknowledgements

The authors thank the KAUST Supercomputing Laboratory for their computing resources.

References

Chapman, B., Curtis, T., Pophale, S., Poole, S., Kuehn, J., Koelbel, C., & Smith, L. (2010). Introducing OpenSHMEM: SHMEM for the PGAS community. *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*, 1–3. https://doi.org/10.1145/2020373.2020375

Dalcin, L., & Fang, Y.-L. L. (2021). mpi4py: Status update after 12 years of development. *Computing in Science & Engineering*, *23*(4), 47–54. https://doi.org/10.1109/MCSE.2021.3083216

Hammond, J. R., Ghosh, S., & Chapman, B. M. (2014). Implementing OpenSHMEM using MPI-3 one-sided communication. *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools: First Workshop, OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014. Proceedings 1*, 44–58. https://doi.org/10.1007/978-3-319-05215-1

Hsu, C.-H., Imam, N., Langer, A., Potluri, S., & Newburn, C. J. (2020). An initial assessment of NVSHMEM for high performance computing. *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 1–10. https://doi.org/10.1109/IPDPSW50202.2020.00104

Jose, J., Zhang, J., Venkatesh, A., Potluri, S., & Panda, D. K. (2014). A comprehensive performance evaluation of OpenSHMEM libraries on InfiniBand clusters. *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools: First Workshop,* *OpenSHMEM 2014, Annapolis, MD, USA, March 4-6, 2014. Proceedings 1*, 14–28. https://doi.org/10.1007/978-3-319-05215-1_2

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., & Stoica, I. (2018). Ray: A distributed framework for emerging AI applications. *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 561–577. ISBN: 978-1-939133-08-3

Potluri, S., Goswami, A., Rossetti, D., Newburn, C. J., Venkata, M. G., & Imam, N. (2017). GPU-centric communication on NVIDIA GPU clusters with InfiniBand: A case study with OpenSHMEM. *2017 IEEE 24th International Conference on High Performance Computing (HiPC)*, 253–262. https://doi.org/10.1109/HiPC.2017.00037

Rigo, A., & Fijalkowski, M. (2022). CFFI: C Foreign Function Interface for Python. In *Read the Docs*. https://cffi.readthedocs.io/

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. *Proceedings of the 14th Python in Science Conference*, *130*, 136. https://doi.org/10.25080/Majora-7b98e3ed-013

Rogowski, M., Aseeri, S., Keyes, D., & Dalcin, L. (2023). mpi4py.futures: MPI-based asynchronous task execution for Python. *IEEE Transactions on Parallel and Distributed Systems*, *34*(2), 611–622. https://doi.org/10.1109/TPDS.2022.3225481

Si, M., Peña, A. J., Hammond, J., Balaji, P., Takagi, M., & Ishikawa, Y. (2015). Casper: An asynchronous progress model for MPI RMA on many-core architectures. *2015 IEEE International Parallel and Distributed Processing Symposium*, 665–676. https://doi.org/10.1109/IPDPS.2015.35