An Initial Implementation of Libfabric Conduit for OpenSHMEM-X*

Subhadeep Bhattacharya[1], Shaeke Salman[1], Manjunath Gorentla Venkata[2], Harsh Kundnani[1], Neena Imam[2], and Weikuan Yu[1]

[1] Department of Computer Science
Florida State University
{bhattach,salman,kundnani,yuw}@cs.fsu.edu
[2] Oak Ridge National Laboratory
{manjugv,imamn}@ornl.edu

Abstract. As a representative of Partitioned Global Address Space models, OpenSHMEM provides a variety of functionalities including one-sided communication, atomic operations, and collective routines. The communication layer of OpenSHMEM-X plays a crucial role for its functionalities. OFI Libfabric is an open-source network library that supports portable low-latency interfaces from different fabric providers while minimizing the semantic gap across API endpoints. In this paper, we present the design and implementation of OpenSHMEM-X communication conduit using Libfabric. This Libfabric conduit is designed to support a broad range of network providers while achieving excellent network performance and scalability. We have performed an extensive set of experiments to validate the performance of our implementation, and compared with the Sandia OpenSHMEM implementation. Our results show that the Libfabric conduit improves the communication bandwidth on the socket provider by up to 42% and 11%, compared to an alternative OpenSHMEM implementation for put and get operations, respectively. In addition, our implementation of atomic operations has achieved similar latency to that of the Sandia implementation.

1 Introduction

SHMEM is known as a collection of programming libraries that provide parallel processing capabilities. Cray systems developed the first implementation. Other organizations later followed suit with their own implementations, including SGI, IBM and many

* This work was sponsored by the U.S. Department of Energy's Office of Advanced Scientific Computing Research. This manuscript has been authored 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). 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.

others. One-sided communication, shared view of memory, atomic and collective operations are some critical aspects of SHMEM library implementation. It has been used in the context of parallel programming model to implement partitioned global address space (PGAS) systems.

The presence of various implementations of SHMEM prompted the need of a standard specification and API. In addition, performance and code portability is also critical due to diverged behaviors and API syntax from different implementations of SHMEM. The OpenSHMEM community was formed to standardize the specification, and they introduced OpenSHMEM [3], integrating different implementation efforts of SHMEM.

PGAS (Partitioned Global Address Space) languages are popular due to their capability of providing shared memory programming model over distributed memory machines. OpenSHMEM [9] realizes a PGAS model by allocating remotely accessible objects and enabling easy data sharing among the processing elements (PEs) in an application. It can be leveraged in developing parallel, scalable and portable programs, achieving low latency and high bandwidth. OpenSHMEM data distribution is carried out by one-sided communication and synchronization routines. Such one-sided operations allow a local PE to be independent of the communication at a remote PE when executing and completing a data transfer operation. It also facilitates a simplified exhibition of parallelism allowing overlap between communication and computation that hide data transfer latencies.

Libfabric [6] [8] is an open-source software library for various network fabrics. It is implemented by the OpenFabrics Interface (OFI) working group. Libfabric aims to reduce the technical complexities between applications and underlying fabric services and support many application semantics directly. It has been carefully designed by fabric hardware providers and application developers, so that HPC users can get the utmost benefit. Libfabric is agnostic to the underlying hardware implementation and networking protocols. Its goal is to utilize high-bandwidth and low-latency network interface cards (NICs) to deliver a highly scalable and portable network implementation.

In this work, we introduce an implementation of a new Libfabric conduit for OpenSHMEM to achieve high-performance communication. We take on an existing OpenSHMEM implementation, OpenSHMEM-X [10] from ORNL. Currently, OpenSHMEM-X includes a couple of existing network conduits: GASNet [5] and UCX [15], which supporting mapping between OpenSHMEM calls and network operations. UCX [13] consists of a collection of network libraries and interfaces having the capability of rendering high scalability and throughput. UCX maintains the networking paradigms of high-performance computing, allowing RMA operations, tag matching, remote atomic operations and active messages. From the OpenSHMEM perspective, Libfabric and UCX provide functionality similar to GASNet. Like UCX, Libfabric also provides similar kind of APIs for initialization, shutdown, data transfer, atomic operations.

Our objective is to design the Libfabric conduit in OpenSHMEM-X communication layer with the purpose of providing a broad range of network support while enhancing the performance and scalability. We carefully devise the initialization and shutdown of the network layer with the most suitable options provided by Libfabric. PMIx is utilized as the out-of-band channel for exchanging the necessary information among the PEs. We focus more on designing the point-to-point and atomic operations as these pro-

vide the foundation for most of the other OpenSHMEM-X functionalities. Additionally, we utilize the different data transfer operations of Libfabric depending on data size to achieve optimal network performance.

We assess our work in the Eos system of Oak Ridge Leadership Computing Facility and compare it with the Sandia OpenSHMEM implementation. Therefore, we choose Libfabric as the communication layer while configuring both the application. After accumulating the results for point-to-point and atomic data transfer operations, we observe that our implementation outperforms Sandia OpenSHMEM for blocking put, get and atomic operations.

The rest of this paper is organized as follows: In Section 2, we present some backgrounds on OpenSHMEM-X and Libfabric. In Section 3, we describe some works that are related to our implementation. We then elaborate our design of Libfabric conduit in Section 4. Section 5 presents the performance evaluation of our implementation and a comparison with Sandia OpenSHMEM Libfabric implementation. Finally, we discuss possible tuning and feature enhancements in Section 6 and conclude in Section 7.

2 Background

2.1 OpenSHMEM-X

SHMEM programs follow the Single Program Multiple Data (SPMD) parallel programming model where all PEs run the same program. The actual number of PEs is set during the start of the program. They are numbered from 0 to $N-1$, where N stands for the total number of PEs. SHMEM library routines offer remote data transfer, synchronization, collective and atomic memory operations to attain high-performance communication. It creates a symmetric view of memory for the participating PEs, where they can allocate symmetric variables maintaining their copies. This feature allows remote direct memory access (RDMA) between the processing elements. One participating PE can directly put or get data from the particular memory location of another PE. RDMA capable interconnects are particularly attractive to the PGAS community as they provide high-throughput, low-latency networking. Vendors such as Intel, HPE, Cray have provided high-performance OpenSHMEM implementations. In addition to that, there are implementations by ORNL known as OpenSHMEM-X. The current OpenSHMEM-X implementation features GASNet and UCX communication conduits.

2.2 Libfabric

Libfabric is aimed to support low-latency communication on network hardware. Libfabric library is designed to support many different programming models such as MPI, and PGAS over various fabric hardware (such as sockets, InfiniBand, Cray GNI, Intel TrueScale, Intel Omni-Path, and Cisco VIC). The design of libfabric provides a set of standard interfaces to set up the communication environment. Libfabric APIs are developed by different underlying providers, each operating over specific fabric hardware or software abstraction. The standard interface functions need to be implemented by each provider. Each provider can also choose the interface functions they want to support. Moreover, the application has the option to scrutinize the capabilities of providers and accordingly, choose to use the most appropriate providers for communications support.

3 Related Work

For its popularity, there has been a large body of research and development studies on OpenSHMEM. Here we selectively provide a discussion on a few studies. Hammond et al. [7] developed an OpenSHMEM implementation using many new communication features proposed by the MPI-3 community. Particularly, this study emphasized the integration of MPI-3 network capabilities such as remote atomic functionalities and a memory model similar to the model required for an OpenSHMEM implementation. Similar to MPI-3 communication routines, Libfabric supports remote atomic functionalities and an optimized memory management scheme, which makes it a viable choice as the network conduit for OpenSHMEM. To this end, our work represents a new attempt to develop the Libfabric conduit for the OpenSHMEM-X implementation from Oak Ridge National Lab.

New network layers have also been introduced in recent times. It helps to achieve portability and scalability across various networks while providing high-performance for different HPC applications. Shamis et al. [14] introduces the implementation of OpenShmem using Universal Common Communication Substrate (UCCS) to optimize its performance by redesigning the network layer. UCCS is a low-level communication library explicitly designed to implement parallel programming models. It supports atomic and remote memory operations which are required by any PGAS programming model. UCCS implementation in OpenSHMEM shows that it outperforms state of the art SGI's SHMEM. Baker et al. [1] has developed an implementation of OpenSHMEM on UCX, validated its suitability for OpenSHMEM programming model, evaluated its performance and scalability on Cray XK systems. UCX have a close semantic match with OpenSHMEM APIs and provides portability and scalability to HPC applications. Built on top of these prior implementations for various SHMEM frameworks, our implementation extends the OpenSHMEM-X implementation with an additional conduit that supports the Libfabric library.

Seager et al. [12] have developed the Sandia OpenSHMEM implementations using Libfabric, which is the closest to our work for OpenSHMEM-X. They have demonstrated the outstanding improvement in bandwidth when OFI Libfabric is used as a transport layer. The Sandia implementation has been enabled on the Aries interconnect that uses u-GNI as the low-level software interface. Compared to the Cray SHMEM that uses DMAPP instead of u-GNI, they have achieved a significant bandwidth improvement. Moreover, Sandia OpenSHMEM uses the message injection feature of Libfabric to enable high bandwidth for small messages. Compared to Sandia OpenSHMEM, our work enables a Libfabric conduit for the OpenSHMEM-X library. Similar to Sandia OpenSHMEM, we also utilize the message injection to get higher performance characteristics for small messages and atomic operations.

Overall, we have designed the OpenSHMEM-X implementation using salient features available from Libfabric to enable efficient, scalable and portable communication.

4 Design of Libfabric Conduit

In this section, we describe the design of Libfabric conduit as an additional communication path for OpenSHMEM-X. We will focus on the description of initialization,

out-of-band configuration, memory management, address translation and communication routines.

4.1 Architectural Overview of Libfabric Conduit

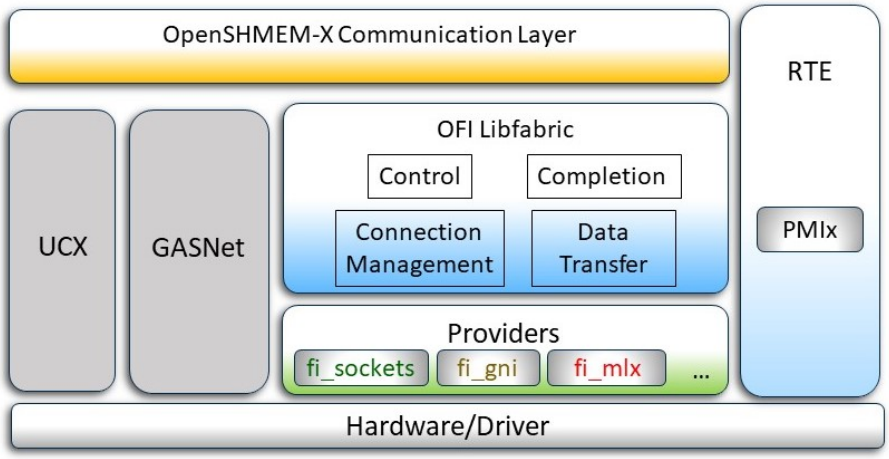


Fig. 1: Software Architecture of OpenSHMEM-X with Libfabric Conduit

OpenSHMEM-X provides a dedicated layer to integrate any network conduit for communication. The communication layer is designed in a way that common functionalities can be shared across different network conduits. Thus, any implementation of an additional conduit needs to take a close architectural examination and make careful choices to design a thin yet high-performance network path.

As a communication library, Libfabric delivers its portability across many popular network providers. It hides the complexity of network management and data transfer inside its implementation, and offers a thin-layer of APIs for application performance and scalability. Fig. 1 depicts the software architecture of OpenSHMEM-X with particular details on its communication functionalities. Besides the existing UCX and GASNet, we extend OpenSHMEM-X with Libfabric conduit. We plan to organize the communication architecture of the conduit into four major components including connection management, control functionalities, data transfer operations and completion events. PMIx [2] [11] is used as the RTE layer that enables out-of-band communication. To avoid redundancy, we leverage useful data structures from the OpenSHMEM-X and Libfabric as much as possible.

We elaborate the details of communication functionalities on remote memory access routines, atomic operations and point-to-point synchronization operations in the rest of this section.

4.2 Initialization and Out-of-Band Configuration

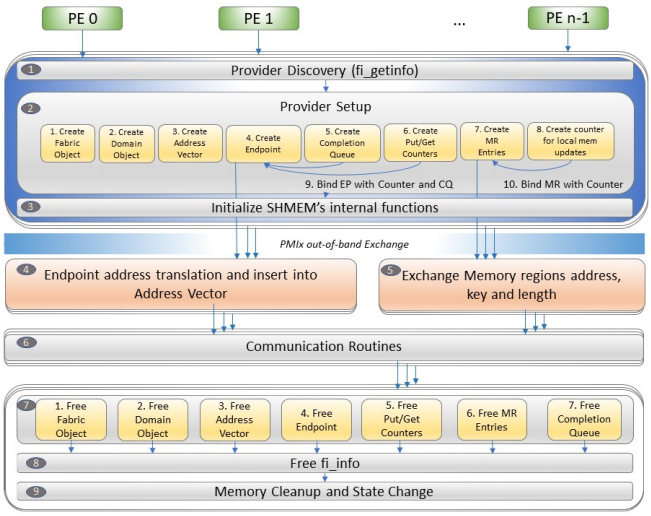


Fig. 2: Initialization and Shutdown of Libfabric Conduit in OpenSHMEM-X

The primary responsibility while implementing any scalable network layer is to configure resources and initialize their parameters before any communication, and properly shut down the resources in the end. Fig. 2 describes the initialization and shutdown of Libfabric conduit in an OpenSHMEM-X program. We leverage the control operations from Libfabric for handling these functionalities. The capabilities for a Libfabric provider can be obtained via an initial query to the underlying provider library shown in Step 1. The discovery API then fetches a set of fabric interfaces with their capabilities based on the requirements specified in the query.

As shown in Step 2 of Fig. 2, our conduit proceeds to query the fabric domain, determining access attributes and endpoint configurations. Libfabric can be utilized in a system containing multiple hardware and software resources for network operations. Hence, it is necessary to query the fabric domain after the provider discovery has identified available resources for communication. After the determination of a fabric domain, an access domain for communication also needs to be determined for a single logical connection to the fabric. Then active endpoints will be configured for data transfer operations. Detailed parameters for the access domain are configured. Furthermore, we will initialize the memory regions and address vectors for each PE before enabling the communication endpoints.

Note that the out-of-band communication layer plays a significant role in OpenSHMEM-X because the addresses of connection endpoints and memory regions for each PE need to be exchanged among all the PEs. Steps 4 and 5 in Fig. 2 describe the steps of address translation and exchange of memory region information. We leverage the PMIx

(**P**rocess **M**anagement **I**nterface E**x**ascale) component for out-of-band communication in our implementation. Step 6 represents the communication routines which is described in the following subsections in details. Finally, in Steps 7, 8 and 9, we free the resources before shutting down the conduit.

4.3 Memory Management and Address Translation

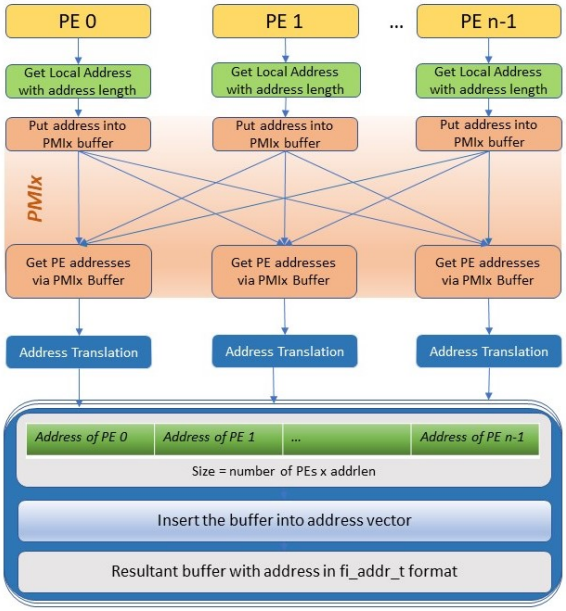


Fig. 3: Address Translation using PMIx

OpenSHMEM provides symmetric memory so that data movement can be performed directly via a memory address to any location in the Heap, BSS, and Data sections. Thus all these memory regions need to be registered via the communication conduit before the one-sided operations. The information on the Heap, Data and BSS segments shall be made available via their start addresses and range. In the current implementation of OpenSHMEM-X, basic memory registration is first performed for these sections by the PE. Then the start address, length and key of all registered memory regions are to be extracted. An array of IO vectors (*fi_rma_iov*) from Libfabric is used to record the memory region information for each PE. All the PEs then exchange this array through the PMIx out-of-band communication channel. For the exchange of these vectors at the beginning of an OpenSHMEM program, fabric specific addresses are used for data transfer operations. These addresses are different from the high-level logical addresses.

Fig. 3 shows the exchange of these vectors through PMIx and the steps involved in address translation. Libfabric provides address vectors, which allows the mapping of addresses without expensive address resolution. The source address of each PE is fetched and exchanged among all the PEs using PMIx. After getting endpoint addresses of all the PEs, the addresses are stored in a single buffer linearly using the length of each endpoint address. The address buffer is then used as the translation table for address vectors. Any logical address from a remote PE can be translated to the fabric specific address. For proper indexing, the address vectors from different PEs are linearly ordered according to the ranks of all PEs.

4.4 Completion Queues and Counters

Libfabric point-to-point operations are non-blocking. Thus there must be some form of progress tracking and completion notification mechanism to ensure the completion of an operation which can be properly detected and notified to the pertinent components. Libfabric provides completion queues and counters for this purpose. A completion queue is bound to the endpoint for getting the completion event of any read, write, or atomic operations. Lightweight counters are also used to keep track of put and get operations. An extra counter is bound with the local memory region to keep track of update operations to the local memory region. This additional counter helps to achieve *wait* functionality of OpenSHMEM-X.

4.5 Remote Memory Access Routines and Atomic Operations

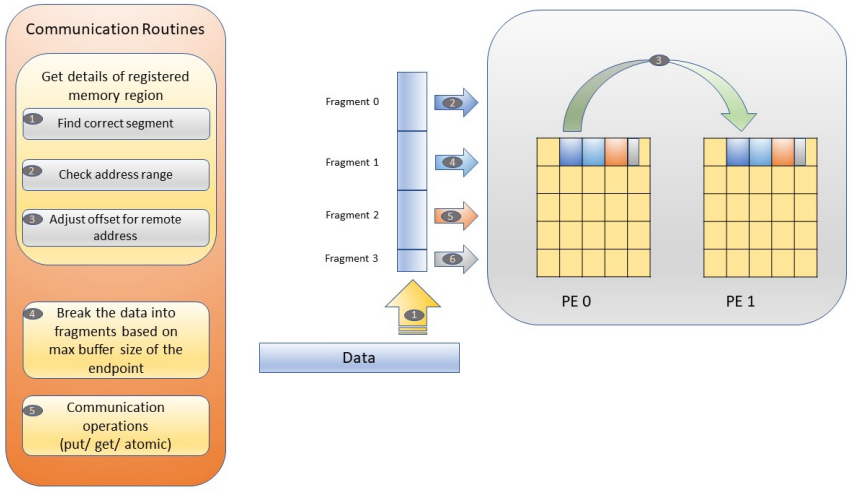


Fig. 4: Data Transfer Operations

OpenSHMEM requires the support of remote memory accesses and atomic operations from the underlying conduit in order to achieve the functionalities of blocking

and non-blocking operations (put, get and atomics). Fig. 4 shows the steps involved in any data transfer operation in OpenSHMEM-X using Libfabric. For any data transfer operation, OpenSHMEM-X first determines the memory segment in the local PE based on the memory address and offset for the remote PE address segment. Once the address segment information including the key and address is available, the data can be written to the remote location using Libfabric APIs.

However, data that need to be written in the remote PE can be of any length. For very large messages, we need to send the data into chunks to avoid overflowing the transmit and receive buffers at the endpoints. The maximum data size that can be sent in each iteration is determined by querying the underlying provider for the maximum supported size of the endpoint. The data is broken into fragments. We iterate the data transfer operations in a loop until the operation on entire data gets completed.

5 Performance Evaluation

In this section, we describe our experimental setup and the evaluation results. We measure the performance of point-to-point data movement and atomic operations across multiple nodes.

5.1 Experimental Setup

We conduct all our experiments on the *Eos* system [4] located at Oak Ridge National Laboratory. Eos is an XC30 computing cluster built by Cray. It is equipped with Intel Xeon E5-2670 CPUs, 16 cores per node. The whole system consists of 736 nodes with a total of 47.104 TB of memory where each node comprises of 64GB memory. Eos uses Cray's Aries interconnect and Dragonfly network topology.

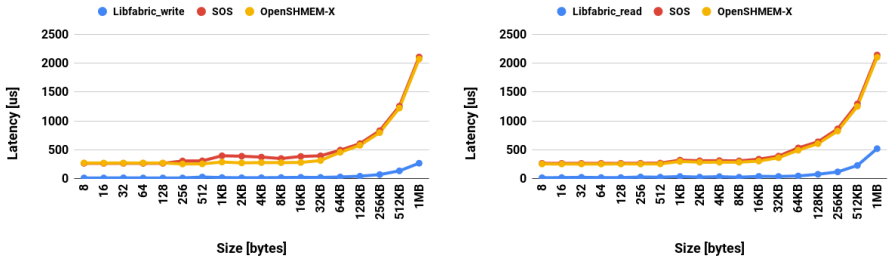
For our measurements, we use the OpenSHMEM-X code base and configure its communication path to use our implementation of Libfabric conduit. In our conduit, the Sockets provider of Libfabric is used for our initial implementation. We also configure Sandia OpenSHMEM with Libfabric on the Eos system, for which the sockets provider is also chosen for Sandia OpenSHMEM (SOS). The Sandia OpenSHMEM is configured with hard and completion polling. For a fair comparison of the Libfabric conduits in the two OpenSHMEM libraries, we also disable the scalable memory registration and multi-threading in Sandia OpenSHMEM, because these functionalities are not implemented in the OpenSHMEM-X Libfabric conduit. We use OpenSHMEM micro-benchmarking suite (SHOMS) to measure the performance of point-to-point operations. OSU Micro-Benchmarks tool is used to measure the latency of atomic operations. Fabtest is used for measuring the Libfabric read and write performances.

5.2 Performance of Point-to-Point Data Movement Operation

We run our tests across multiple nodes in EOS. For point-to-point data movement operations in SHOMS, node 0 sends a fixed sized message to all other nodes and collects the bandwidth. We use the default configuration for SHOMS which use 1000 iterations for data size 8 bytes to 16 KB, 500 iterations for data size of 16 KB to 512 KB and 250

iterations for messages greater than 512 KB. Minimum and maximum message sizes are 8 bytes and 1 MB, respectively. We collected the aggregated bandwidth and average latency measurements for our evaluation. We also compare the bandwidth and latency with Sandia OpenSHMEM implementation and Libfabric read/write operations.

Put Latency Fig. 5(a) shows the comparison of latency measurements between OpenSHMEM-X Libfabric conduit, Sandia OpenSHMEM Libfabric implementation and the unidirectional write operation of Libfabric. SOS performs slightly better by about 2% for the message size upto 128 bytes. However, our implementation shows a better latency measurement for message size greater than 128 bytes with almost 30% improvement for 2 KB messages.



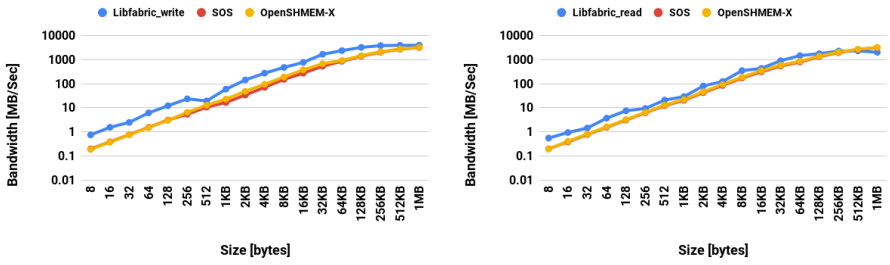
(a) `PUT` latency (b) `GET` latency

Fig. 5: `Latency` Comparisons

Get Latency Fig. 5(b) shows the latency measurements for OpenSHEMEM-X, Sandia OpenSHMEM and Libfabric read operation. Our implementation performs better than SOS in terms of latency. We get a latency improvement up to 10% for 16 KB message size.

Put Bandwidth Fig. 6(a) shows the bandwidth of PUT operation for OpenSHMEM-X Libfabric conduit, Sandia OpenSHMEM Libfabric implementation and unidirectional write bandwidth of Libfabric. The *shmem_int_put* operation is used as the test case for measuring the PUT benchmark. Our evaluation results show that the performance of PUT operations in Sandia OpenSHMEM is 2-5% better than our implementation for small data sizes up to 128 bytes. However, the performance of our implementation improves for messages greater than 256 bytes. The bandwidth improvement can be up to 42% for 2 KB messages.

Get Bandwidth Fig. 6(b) compares the bandwidth of GET operations between OpenSHEMEM-X, Sandia OpenSHMEM and Libfabric read operation. The *shmem_int_get* operation is

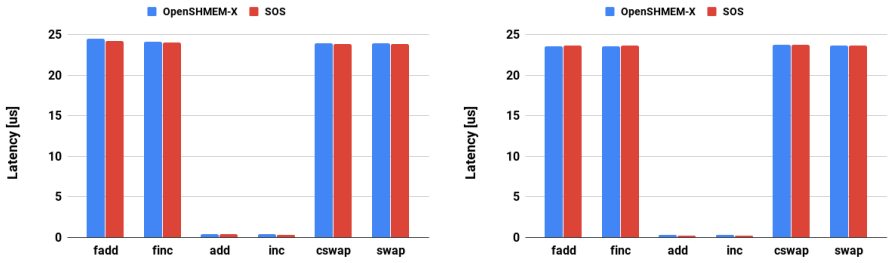


(a) `PUT` bandwidth (b) `GET` bandwidth

Fig. 6: `Bandwidth` Comparisons

used for measurements in this case. Our experimental results demonstrate an improvement up to 11% over Sandia OpenSHMEM implementation.

5.3 Atomic Operations



(a) `Atomic` 32-bit latency (b) `Atomic` 64-bit latency

Fig. 7: Atomic operation performance

Fig. 7(a) and Fig. 7(b) show the atomic operation performances for OpenSHMEM-X and Sandia OpenSHMEM. We collect the latency of atomic operations for 32-bit and 64-bit messages using OSU microbenchmarks. We have collected the performance measurements for six integer atomic operations: Fetch and Add, Fetch and Increment, Add, Increment, Compare and Swap, and Swap. We have run each atomic operation for 1000 iterations and collected the average latency measurements. The performance of these atomic operations from our implementation is comparable with, or slightly better than, the Sandia OpenSHMEM implementation.

6 Discussion

Libfabric offers support for many provider implementations such as verbs, mlx, and gni. We have tried verbs and mlx providers using Libfabric. However, *fi_verbs* and *fi_mlx* both do not support full atomic functionalities required by the OpenSHMEM-X. One of the most common Libfabric providers for Cray systems is uGNI, which is fully supported on ORNL systems. We are still working on enabling the uGNI provider for the OpenSHMEM-X Libfabric conduit. Furthermore, we are going through additional tuning and optimization efforts. When these steps are completed, we expect to see a further performance improvement on the OpenSHMEM-X Libfabric conduit.

7 Conclusion

Libfabric offers a communication library that minimizes the semantic gap and maintains application performance while delivering scalability. Its application-centric design allows us to enable operations on different networking interface without considering the internal hardware management. In this paper, we have designed and prototyped an implementation of Libfabric conduit for the OpenSHMEM-X library from ORNL. We have evaluated the performance of our implementation and compared it with the Sandia OpenSHMEM implementation. Our assessment demonstrates that our Libfabric conduit can indeed enable OpenSHMEM portably on different network providers while achieving excellent performance and scalability. We are working on further tuning and optimizations of our Libfabric conduit. We will also work on enabling more Libfabric providers for OpenSHMEM-X.

Acknowledgment

This work is supported in part by a contract from Oak Ridge National Laboratory and the National Science Foundation awards 1561041 and 1564647.

We are thankful to Amit Kumar Nath for his valuable suggestions and feedbacks to the paper. We would like to thank Matthew B. Baker (ORNL) for his help in figuring out the details of OpenSHMEM-X, and Arun Ilango, Sean Hefty and Sayantan Sur from Intel for their valuable comments and suggestions on solving the technical difficulties regarding OFI Libfabric.

This research was supported by the United States Department of Defense (DoD) and Computational Research and Development Programs at Oak Ridge National Laboratory.

This work was sponsored by the U.S. Department of Energy's Office of Advanced Scientific Computing Research. 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.

References

1. M. Baker, F. Aderholdt, M. G. Venkata, and P. Shamis. Openshmem-ucx: Evaluation of ucx for implementing openshmem programming model. In *OpenSHMEM*, 2016.
2. R. H. Castain, D. Solt, J. Hursey, and A. Bouteiller. PMIx: Process Management for Exascale Environments. In *Proceedings of the 24th European MPI Users' Group Meeting*, EuroMPI '17, pages 14:1–14:10, New York, NY, USA, 2017. ACM.
3. B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, and L. Smith. Introducing openshmem: Shmem for the pgas community. In *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model*, PGAS '10, pages 2:1–2:3, New York, NY, USA, 2010. ACM.
4. EOS, Cray® XC30™. Specifications available at `https://www.olcf.ornl.gov/for-users/system-user-guides/eos/`.
5. GASNet. Available at `https://gasnet.lbl.gov/`.
6. P. Grun, S. Hefty, S. Sur, D. Goodell, R. D. Russell, H. Pritchard, and J. M. Squyres. A brief introduction to the openfabrics interfaces - a new network api for maximizing high performance application efficiency. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 34–39, Aug 2015.
7. J. R. Hammond, S. Ghosh, and B. M. Chapman. Implementing openshmem using mpi-3 one-sided communication. In *Proceedings of the First Workshop on OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools - Volume 8356*, OpenSHMEM 2014, pages 44–58, New York, NY, USA, 2014. Springer-Verlag New York, Inc.
8. Libfabric. `https://ofiwg.github.io/libfabric/`.
9. OpenSHMEM. `http://www.openshmem.org/site/`.
10. OpenSHMEM-X. Available at `https://github.com/ornl-languages/ornl-openshmem.git`.
11. PMIx. Available at `https://pmix.org/`.
12. K. Seager, S.-E. Choi, J. Dinan, H. Pritchard, and S. Sur. Design and implementation of openshmem using ofi on the aries interconnect. In M. Gorentla Venkata, N. Imam, S. Pophale, and T. M. Mintz, editors, *OpenSHMEM and Related Technologies. Enhancing OpenSHMEM for Hybrid Environments*, pages 97–113, Cham, 2016. Springer International Publishing.
13. P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss, Y. Shahar, S. Potluri, D. Rossetti, D. Becker, D. Poole, C. Lamb, S. Kumar, C. Stunkel, G. Bosilca, and A. Bouteiller. Ucx: An open source framework for hpc network apis and beyond. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 40–43, Aug 2015.
14. P. Shamis, M. G. Venkata, S. Poole, A. Welch, and T. Curtis. Designing a high performance openshmem implementation using universal common communication substrate as a communication middleware. In S. Poole, O. Hernandez, and P. Shamis, editors, *OpenSHMEM and Related Technologies. Experiences, Implementations, and Tools*, pages 1–13, Cham, 2014. Springer International Publishing.
15. UCX. Available at `https://www.openucx.org/`.