SAND2016-8434C

SHMEM-MT: A Benchmark Suite for Assessing Multi-threaded SHMEM Performance

Hans Weeks[1], Matthew G. F. Dosanjh[2], Patrick G. Bridges[1], and Ryan E. Grant[2]

[1] Department of Computer Science, University of New Mexico, Albuquerque, USA
{hansel,bridges}@cs.unm.edu
[2] Center for Computing Research, Sandia National Laboratories, Albuquerque, USA
{mdosanj,regrant}@sandia.gov *

1 Introduction

OpenSHMEM is a popular one-sided communication library for high-performance computing systems developed around 2010 at the University of Houston [2]. It is becoming an increasingly popular programming model for next-generation HPC applications and systems because of its simple, intuitive interface and the proliferation of one-sided communication devices such as Infiniband [1]. Despite its increasing popularity, there are few benchmarks or mini-applications for evaluating and optimizing OpenSHMEM system software and hardware performance. This is particularly true for emerging multi-core and many-core systems on which OpenSHMEM is particularly important.

In this paper, we present the first set of OpenSHMEM benchmarks of which we are aware for systematically evaluating OpenSHMEM communication performance. A key element of these benchmarks is their support for multi-threading, based on the OpenSHMEM thread API proposed by Cray [9]. These benchmarks are based on one-sided benchmarks and mini-applications previously developed for MPI [4]. The initial version described in this paper focuses on simple messaging micro-benchmarks and HPC mini-applications, in both cases with simple synchronization strategies; support for additional benchmarks, mini-applications, and synchronization methods is planned.

2 SHMEM-MT Benchmarking Approach

To develop a set of OpenSHMEM benchmarks for driving communication system design and optimization, we have thus far focused on porting the MPI RMA-MT benchmarks [4]. This work primarily focused on identifying the proper way to port MPI RMA one-sided calls to OpenSHMEM, how the benchmarks were

* Sandia National Laboratories s a multiprogram laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

converted from MPI RMA to OpenSHMEM, and how threading was supported when appropriate in the current version of these benchmarks.

The RMA-MT benchmark suite [4] was designed to provide a robust set of tests to verify the functionality and measure the performance of MPI's one-sided communication implementation in a multi-threaded environment. These benchmarks are based on previous benchmarks, including Thakur and Gropp's multi-threaded latency and bandwidth tests [10], the Sandia Microbenchmarks (SMBs) [3], and a subset of the Mantevo Mini-Applications [5]. RMA-MT generally focuses on the most commonly-used subset of MPI one-sided calls. We used the RMA-MT benchmark suite as the basis for the OpenSHMEM benchmarks we present here.

Benchmark Conversion When replacing synchronization methods in our benchmarks, we used the synchronization methods that best matched the communication pattern used by the benchmark. In latency and bandwidth benchmarks, we used `shmem_quiet` because only one processing element need be involved in the communication (passive target). For message-rate and mini-app benchmarks, we used `barrier_all` because the underlying applications already relied on barriers to synchronize the activities of multiple processes. Importantly, we have not yet attempted to port the lock-based versions of the RMA-MT benchmarks for SHMEM-MT because of the significant semantic differences between these communications (active target).

In contrast to the synchronization calls, converting the MPI window management and RMA calls was straightforward. In particular, we replaced the calls to `malloc` and `MPI_win_create` with appropriate `shmem_malloc` calls and replaced MPI_get/put with shmem_get/put. In addition, because command line arguments given to the benchmarks are global variables that are stored in symmetric memory, we were able to remove calls to broadcast these parameters that were present in the original RMA-MT benchmarks.

It is important to note that the mini-applications still use a hybrid MPI/-OpenSHMEM approach in some cases. As with the RMA-MT benchmarks, we focused on converting the main halo exchange of each application to OpenSHMEM to test the performance-critical communications at scale in an application setting. Other communications such as set-up and tear-down, as well as a handful (one or two) `MPI_Allreduce` calls per iteration in each mini-application, are still performed using MPI. This approach is similar to that taken by other researchers [6, 7]. We hope to convert the remaining MPI communication operations in these mini-applications to OpenSHMEM in the near future, but have not prioritized this effort as these calls are not generally performance critical at the scale at which we currently execute.

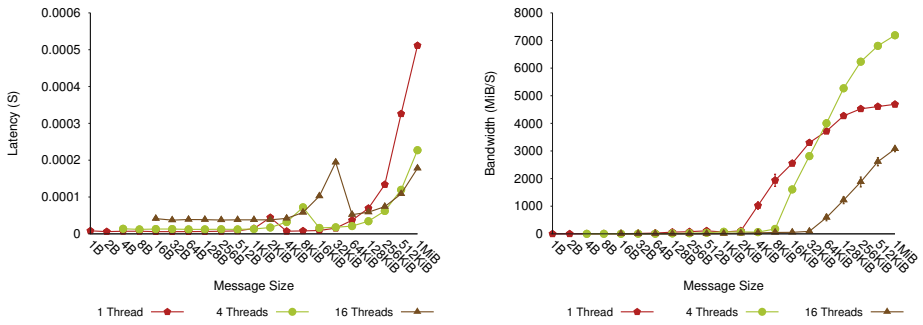
Threading Support Because MPI works on a per-process basis, the RMA-MT messaging benchmarks rely on per-process synchronization and use threads only for RMA data movement operations. In particular, these benchmarks multi-thread operations between synchronization calls using using a fork-join threading

model. The benchmarks use this structure primarily because of the lack of fine-grained thread-level synchronization operations in current versions of MPI, as threads are not separate entities recognized by MPI, unlike in the OpenSHMEM thread extensions provided by Cray.

Our initial port of the RMA-MT benchmarks to OpenSHMEM preserves the basic fork-join threading structure of these benchmarks that results from the lack of thread-level synchronization primitives in MPI RMA calls. In particular, we preserved synchronization methods at the processing element granularity by calling `shmem_quiet` or `shmem_barrier_all` after `pthread_join` at the end of each test iteration, rather than relying on thread-level synchronization. Converting these benchmarks to use the thread-level synchronization primitives proposed for use in OpenSHMEM, for example `shmem_thread_quiet` and `shmem_thread_fence` is an important direction for future work.

3 Initial Results

In this section we present initial results using this benchmark suite on a Cray XC30 cluster. Each node has two Xeon Ivy Bridge 2.4 GHz 12-core processor with hyper-threading enabled, 32 GB of memory per node, and a Cray Aries network interface. SHMEM-MT benchmarks were compiled using the Cray compiler suite and Cray shmem version 7.3.2. Each data point in this section is an average of 10 runs with each run performing 10,000 iterations, in the case of the messaging benchmarks. Each point is plotted with error bars showing the standard deviation of the 10 runs; in a large number of cases, the standard deviation of the ten runs was small enough not to show up on the plots.



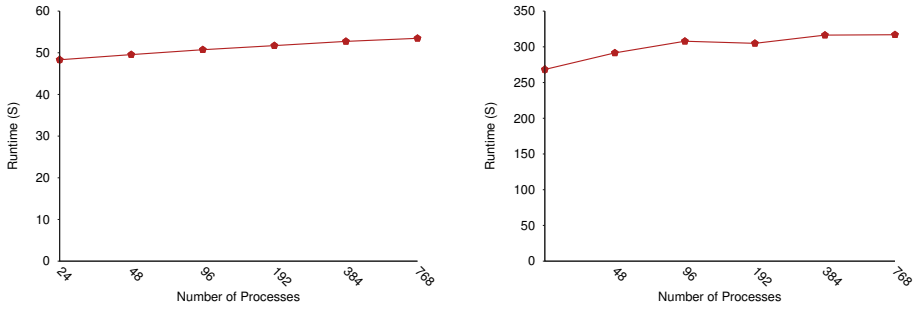
(a) Latency (b) Bandwidth

Fig. 1. SHMEM-MT Latency and Bandwidth Performance

Figure 1 shows the latency and bandwidth tests respectively. Both tests are setup similarly; as the number of threads increase each message is split into smaller equal pieces for each thread. We can see that for small messages, less than

32KiB, Cray SHMEM achieves best performance when using a single thread. After 32 KiB, 4 threads sending portions of the message appear to outperform the 1 thread case. For the message sizes used in this test, 16 threads performs worse in bandwidth than the other cases, likely due to insufficient hardware-level concurrency to amortize the increased synchronization overheads.

Figure 2 shows the runtime of the HPCCG and MiniFE mini-applications when run with 24 ranks per node on up to 32 nodes using a weak scaling problem size. In particular, HPCCG was set to 100^3 elements per PE while the MiniFE problem size was set at $(330 * nodes^{1/3})^3$. Note that, these mini-applications do not yet include full threading support. In this case, messaging concurrency is provided at the thread level, however we run a PE per core to maintain computational concurrency.



(a) HPCCG (b) MiniFE

Fig. 2. SHMEM-MT Mini-application Weak-scaling Runtime

In both cases, performance is largely constant as expected, particularly for MiniFE. HPCCG runtimes begin to increase slightly with increased scale, but to a level that is generally expected for this mini-application. Note that both mini-applications also include solution verifications provided from the original Mantevo versions that complete successfully.

4 Related Work

The most relevant related work to the work presented here is work by Luecke et al. [8] where they compared the performance of SHMEM with MPI-2 RMA on an SGI Origin 2000 and Cray T3E system. Unlike this work, they used a single threaded approach and the MPI-2 RMA interface was the only one available. Since that time, significant improvements have been made to the MPI RMA interfaces for MPI-3, and OpenSHMEM [2] has emerged as a standard, with matching implementations.

5 Conclusions and Future Work

Overall, our work provides a first set of benchmarks for evaluating OpenSHMEM implementations and optimizations, particularly in the presence of multiple threads. Our initial results show OpenSHMEM performance that is generally comparable to other modern messaging systems; due to time and space limitations, we defer a complete performance comparison across MPI, OpenSHMEM, and similar messaging system implementations on different platforms for future work. In addition, the relative ease with which we converted MPI RMA benchmarks to OpenSHMEM demonstrates a path for developing further benchmarks and mini-applications.

References

1. I. T. Association. *InfiniBand Architecture Specification: Release 1.0*. InfiniBand Trade Association, 2000.
2. 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*, page 2. ACM, 2010.
3. D. Doefler and B. W. Barrett. Sandia MPI microbenchmark suite (SMB). Technical report, Sandia National Laboratories, 2009.
4. M. G. Dosanjh, T. Groves, R. E. Grant, R. Brightwell, and P. G. Bridges. RMA-MT: A benchmark suite for assessing mpi multi-threaded rma performance. In *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (IEEE/ACM CCGrid 2016)*, 2016.
5. M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving performance via mini-applications. *Sandia National Laboratories, Tech. Rep*, 2009.
6. J. Jose, S. Potluri, K. Tomko, and D. K. Panda. Designing scalable graph500 benchmark with hybrid MPI+OpenSHMEM programming models. In *Supercomputing*, pages 109–124. Springer, 2013.
7. M. Li, J. Lin, X. Lu, K. Hamidouche, K. Tomko, and D. K. Panda. Scalable MiniMD design with hybrid MPI and OpenSHMEM. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, page 24. ACM, 2014.
8. G. R. Luecke, S. Spanoyannis, and M. Kraeva. The performance and scalability of SHMEM and MPI-2 one-sided routines on a SGI Origin 2000 and a Cray T3E-600. *Concurrency and Computation: Practice and Experience*, 16(10):1037–1060, 2004.
9. M. ten Bruggencate, D. Roweth, and S. Oyanagi. Thread-safe SHMEM extensions. In *Workshop on OpenSHMEM and Related Technologies*, pages 178–185. Springer, 2014.
10. R. Thakur and W. Gropp. Test suite for evaluating performance of MPI implementations that support MPI_THREAD_MULTIPLE. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 46–55. Springer, 2007.