The impact of process topology on RMA programming models: A study on NERSC Perlmutter

Nikodemos Koutsoheras,
Abhinav Bhatele
University of Maryland, College Park
nikos@umd.edu,bhatele@cs.umd.edu

Sayan Ghosh, Joshua Suetterlein,
Nathan R. Tallent
Pacific Northwest National Laboratory
{sayan.ghosh,joshua.suetterlein,nathan.tallent}@pnnl.gov

Abstract

Contemporary communication latency benchmarks usually measure the performance of pairwise exchanges without considering specific process topology, which demonstrates little variation from the mean behavior since the underlying communication pattern is sufficiently benign for modern networks. Distributed-memory applications often have to deal with overlapped data distributions, which governs the underlying topology of the processes, involving an irregular mix of intranode and internode transfers. One-sided communication or Remote Memory Access (RMA) interfaces can mitigate this irregularity by separating the asynchronous communication and synchronization, exposing remote memory access features via one-sided communication semantics to a global address space.

Performance of the most popular asynchronous RMA interfaces like MPI RMA and SHMEM has steadily improved over the past years due to better support from the hardware vendors and community-driven programming model standardization efforts. Previously on HPE™ Cray™ platforms, Partitioned Global Address Space (PGAS) models such as SHMEM were known to outperform MPI RMA, however, currently we observe better or competitive performance of MPI RMA as compared to SHMEM on NERSC Perlmutter supercomputer.

In this work, we discuss the performance of SHMEM and MPI RMA (comparing with MPI point-to-point) for grid and graph process topologies on NERSC Perlmutter via average and 99th percentile latencies. Our findings indicate that RMA exhibits up to 2.5× better performance for medium to large data sizes as compared to point-to-point for diverse process topologies.

1 Generating Process Topologies

Process topology It is the logical arrangement of processes as per application data distribution in the form of a cartesian multi-dimensional grid or a graph (a process is represented as a "node", with lines or edges connecting the nodes). While the number of (process) neighbors for a particular (process) node remains fixed for a process grid topology, contrasting a process graph topology where nodes may have different set of neighbors. As such, communication involving a process and disproportionate number of neighbors can lead to load imbalance and adversarial one-to-many communication patterns.

Our approach To control the layout of the underlying process topology, we rely on synthetically generated data; particularly generating a Random Geometric Graph (RGG) across the processes in parallel [2], such that for the baseline topology every process shares data with its two adjacent processes, depicting a 1D stencil (grid) pattern. The baseline topology can be modified by adding a customizable amount of "cross" edges to the original RGG, increasing the overall degree of the neighbors in the process graph. Our approach is shown in Fig. 1, we provide options to extract the process topology corresponding to the original RGG as a binary file to be processed later by benchmarks, thereby saving significant memory and computation requirements for generating an RGG. We perform data transfers between process neighbors in the topology graph, shown below.

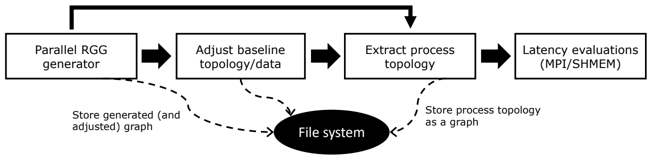
Parallel RGG generator
Adjust baseline topology/data
Extract process topology
Latency evaluations (MPI/SHMEM)
Store generated (and adjusted) graph
File system
Store process topology as a graph

Figure 1: **Our approach for considering process topologies.**

Algorithm 1 Test for data transfer latencies under process graph topology.
Input: $G = (V, E)$, (undirected) process topology graph, G.

```
1: for $s = 1$, $s *= 2$, while $s <= 4MiB$ do
2:     for $u \in \mathrm{adj}(my\_rank)$ do {my process neighbors}
3:         async put $s$ bytes to rank $u$
4:         sync {complete outstanding communication}
```

Existing benchmarks Current MPI and SHMEM benchmarking efforts rarely considers the impact of irregular process topologies, which play a vital role in determining sustainable performance for communication-bound applications. We focus on adapting the underlying process layout from fixed grid to dynamic graph topologies, allowing users the flexibility to measure the performance (following the best practices of reproducible MPI benchmarking [4]) of one-sided community standardized interfaces (MPI RMA and SHMEM), which was originally designed for expressing irregular communication scenarios.

2 Evaluations

Platform details We use the CPU partition of the NERSC Perlmutter supercomputer [9] (Cray™ Shasta architecture), which includes four-way 2.45GHz AMD Milan EPYC™ 7763 CPU nodes comprising of 64 cores, 256MB L3 cache, 512GB DDR4 memory with HPE™ Slingshot interconnect fabric (based on Dragonfly topology) connected to Cassini NIC. Recent study has reported Slingshot interconnection network to be less susceptible to congestion as compared to the previous networks [1, 7]. We use GCC 11.2 compiler (PrgEnv-gnu/8.3.3), cray-mpich/8.2.25 (MPI) and cray-openshmemx/11.5.6 (SHMEM) for the evaluations. Both SHMEM and MPI RMA uses OFI libfabric [3] as the low-level network API. We compare our results

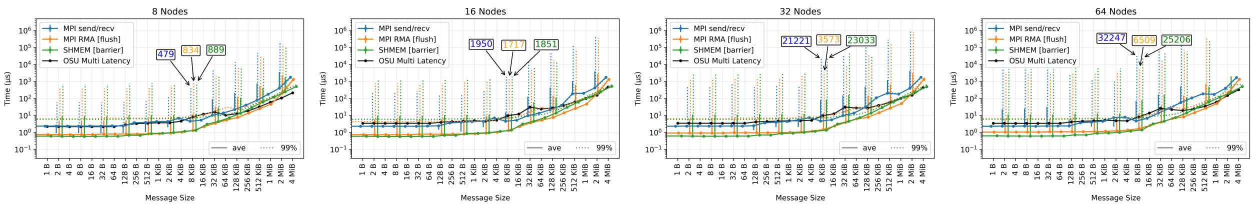

Figure 2: **Avg. and 99th% latencies for 8–64 nodes (16 processes/node) with +2% and +5% extra edges.**

with OSU Microbenchmarks suite [5] v7.6.2 (OSU multi-latency, measures average latencies between multiple process pairs). We use the Perlmutter all-flash scratch file system (aggregate bandwidth >5TB/s) to access the input binary process topology files, overall file I/O is <1.5% of the time to run a test. Our code is available in the following GitHub repository: https://github.com/sg0/neve/tree/rma.

Tests The test kernels comprise of communication functions (like `MPI_Put`, `shmem_char_put_nbi` and `MPI_Isend`/`MPI_Irecv`) and synchronization statements (time includes both, with repetitions and ensuring cold cache), following the pattern in pseudocode 1. We use passive mode synchronization for MPI RMA, using `MPI_Win_flush_all` for synchronization; whereas we use `shmem_quiet` for waiting on outstanding one-sided communication on symmetric heap for SHMEM. For the MPI point-to-point case, we use nonblocking send/receive, with `MPI_Waitall` for completing outstanding request handles. We use bytes as the unit of data transfer, and for every data size calculate average and 99th percentile execution times.

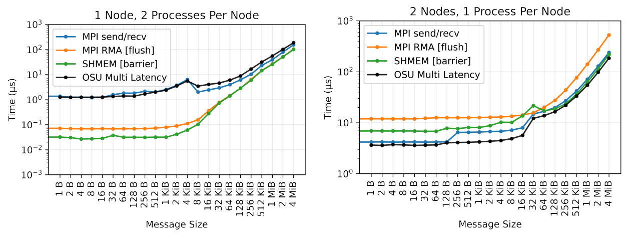

Figure 3: **Avg. latencies for 2-processes within and across nodes.**

Latency measurements We use regular process topology where a process communicates with at most two neighbors, and then consider two more configurations to expand the #edges in the process graph topology by increasing the number of cross edges in the original RGG by 2% and 5% of the overall #edges. As shown in Table 2, just by adding an extra 2% of the overall edges in an input baseline graph, it is possible to make the induced process graph up to about 300× dense, affecting the average and 99th% latencies associated with the data movement.

Table 1: **MPI/SHMEM instructions counts on NERSC Perlmutter (instructions collected on one process using Intel SDE [6]).**

functions	#instructions
shmem_char_put_nbi	192
MPI_Put	231
MPI_Isend	296
MPI_Irecv	204

Table 2: **Baseline and updated process graph topologies.**

nodes /#vertices	#edges		
	base.	+2%	+5%
1(16)	30	240	240
2(32)	62	992	992
4(64)	126	4,032	4,032
8(128)	254	16,224	16,256
16(256)	510	62,468	65,252
32(512)	1,022	210,340	257,340
64(1,024)	2,046	595,060	918,492

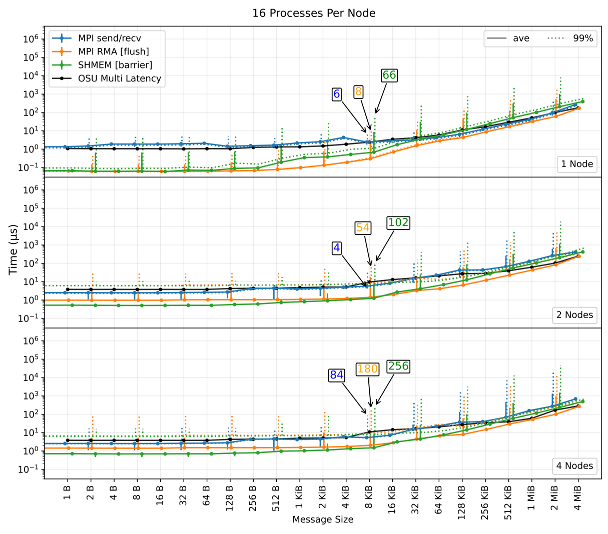

Figure 4: **Avg. and 99th % latencies for 1–4 nodes (16 processes/node).**

Observations Our key takeaways are as follows:

- Two processes measurements (Fig. 3) are not indicative of performance differences at scale when there are more processes/node (point-to-point performance exceeds RMA for a single process/ node, but we see opposite trends with more processes/node).
- OSU microbenchmark reports best case latencies with little variability, whereas the gap between average and 99th percentile latencies surge to orders of magnitude with increasing the node counts (see annotations captured in Fig. 2 vs. Fig. 4).
- For regular topologies on 1–64 nodes, both MPI RMA and SHMEM exceed the performance of point-to-point by up to 3.5×; for irregular topologies, RMA exceeds the performance of point-to-point by up to about 2× for medium to large data (Figures 4 and 2).
- Both SHMEM and MPI RMA demonstrate the best sub-1μs latency for small data sizes considering regular topology. On a single node (Fig. 4), SHMEM depicts 30% better performance up to 8KiB (#instructions in SHMEM are less than MPI, Table 1), but MPI RMA demonstrates better performance by up to 1.5× for larger data sizes.
- As per cray-mpich manual, MPI RMA on NERSC Perlmutter should use XPMEM [8] for single node large message transfers; but, our emulation instructions (via Intel SDE [6]) does not find any entry for XPMEM APIs. On the other hand, SHMEM emulation reveals XPMEM usage.
- On 8–64 #nodes (Fig. 2), SHMEM can exceed the performance of MPI RMA by up to about 2.5× for data sizes beyond 32KiB.

References

[1] Abhinav Bhatele, Kathryn Mohror, Steven H. Langer, and Katherine E. Isaacs. There goes the neighborhood: performance degradation due to nearby jobs. In *Proceedings of the ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '13. IEEE Computer Society, November 2013. URL http://doi.acm.org/10.1145/2503210.2503247.

[2] Sayan Ghosh, Nathan R Tallent, and Mahantesh Halappanavar. Characterizing performance of graph neighborhood communication patterns. *IEEE Transactions on Parallel and Distributed Systems*, 33(4):915–928, 2021.

[3] Paul Grun, Sean Hefty, Sayantan Sur, David Goodell, Robert D Russell, Howard Pritchard, and Jeffrey 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. IEEE, 2015.

[4] Sascha Hunold and Alexandra Carpen-Amarie. Reproducible mpi benchmarking is still not as easy as you think. *IEEE Transactions on Parallel and Distributed Systems*, 27(12):3617–3630, 2016.

[5] Dhabaleswar K Panda et al. Osu micro-benchmarks, 2018.

[6] K Raman. Calculating "flop" using intel software developmentemulator (intelsde)(march 2015).

[7] Daniele Sensi, Salvatore Girolamo, Kim McMahon, Duncan Roweth, and Torsten Hoefler. An in-depth analysis of the slingshot interconnect. in 2020 sc20: International conference for high performance computing, networking, storage and analysis (sc). *IEEE Computer Society*, pages 481–494, 2020.

[8] Michael Woodacre, Derek Robb, Dean Roe, and Karl Feind. The sgi altixtm 3000 global sharedmemory architecture. *Silicon Graphics, Inc*, 44, 2005.

[9] Charlene Yang and Jack Deslippe. Accelerate science on perlmutter with nersc. *Bulletin of the American Physical Society*, 65, 2020.