SANDIA REPORT
SAND2019-11403
Printed Spetember 2019

Receive-Side Partitioned Communication

Matthew G.F. Dosanjh, Ryan E. Grant

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@osti.gov
Online ordering:	http://www.osti.gov/scitech

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.gov
Online order:	https://classic.ntis.gov/help/order-methods

ABSTRACT

This report describes the implementation and experimentation of receive-side partitioned communication for the Message Passing Interface (MPI).

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

The Message Passing Interface (MPI) [36] had included a multi-thread interface for several specification revisions stretching back more than two decades. Although threads have been supported for a long time, MPI multithreading is not widely used today, even if there is interest in using such a programming model [5]. One reason for this lack of adoption is that MPI multithreading support was poorly optimized in some common implementations until recently. Consistent performance between implementations also prevented widespread adoption, and this deficiency is understandable: making a highly performant multithreaded MPI library is complex and challenging. Many hybrid codes today do not directly interact on a thread basis with MPI; instead they marshal threads to ensure a single thread interacts with MPI.

The MPI threading model treats threads in much the same way as processes. Partitioned communication seeks to provide a useful thread-aware MPI interface that offers potential performance improvements and opportunities to increase efficiency through increased computation-communication overlap. Performance improvements could be realized by leveraging thread behaviors and isolating the portion of the MPI API that needs to be thread-safe.

Our previous work in this area with partitioned communication [24, 25] demonstrated that applications can benefit from early-bird overlap of data being sent from threads immediately upon availability rather than waiting to compose an entire MPI message from multiple threads. It demonstrated that there were performance benefits over using existing optimized multi-thread interfaces in MPI that send independent messages from each thread. However, these benefits were only explored for send-side partitioning. We did not explore the potential extra compute time that could be exploited if partitions of a message were exposed to the receiving process to enable early computation on data that had arrived. Our previous partitioned communication work required that data on the receive-side was only exposed when all of the message buffer had arrived.

This work explores the benefits of allowing access to message partitions on the receiver-side to be accessed before the entire message is complete. This is expected to allow for additional time to perform compute with threads/tasks that do not require the data from the entire message to progress. We explore the overheads of supporting receive-side partitioned communication as well as the potential computational time that can be exploited. We find that the exploitable time in typical system noise environments can be significant when compared to the communication time as a whole.

2. BACKGROUND

MPI is message-centric, as its name implies. However, there are overheads to processing messages including their matching stage, placement and message tracking/completion requirements. Typically, this has not been a large concern with single-threaded MPI as impacts of these overheads can be greatly mitigated by taking advantage of expected message ordering and message volume [33, 20, 21].

Multi-threaded MPI had higher overheads than single-threaded MPI as was shown using early benchmarks [40]. Further work in this area showed that message ordering in multi-threaded MPI could be one of the major sources of this higher overhead [37]. As such, researchers have been developing methods of improving matching performance using the existing MPI interfaces for

some time including new data matching structures [41], methods [22, 17, 23] and ways to improve message matching threading locks [16]. In addition, multi-threaded RMA was assessed [18] and improved [28].
While many efforts were made to mitigate the impact of message processing overhead in multi-threaded codes, an alternative is to support multi-threading more explicitly through dedicated interfaces. A recent attempt at this was called MPI endpoints [14]. Endpoints assigned a MPI rank to threads, allowing for messages to originate and complete at specific thread contexts. Unfortunately, endpoints was not adopted by the MPI specifications body as it was determined that many of the benefits of such an approach did not need an entirely new interface in MPI.
Partitioned communication takes a different approach to provide new capabilities and optimization opportunities via a thread-specific interface. Instead of promoting threads to MPI rank level, partitioned communication allows threads to inform MPI that portions of a message buffer are available. This can enable an MPI implementation to move parts of a message to a pre-negotiated buffer on the receiver-side before all of the data becomes available. This early-bird communication allows communication to complete earlier, especially when a small number of threads are lagging behind in processing due to different tasks they are performing or general system noise.
Taking advantage of a spread in thread completion times to send data is one aspect of optimizing partitioned communication. Another aspect that we will explore here is taking advantage of the partitioned nature of the communication to avoid having to fully complete a message before parts of that data buffer can be accessed at the receiver. This allows some computation to occur at the receiver before all threads/tasks that require the contents of the message can proceed, allowing early access for some threads/tasks. This approach can utilize underlying RDMA capabilities in networks to move data efficiently. While it is known that doing so while running applications can impact performance [26] there do exist strategies for completely mitigating such impacts [27].

3. EXPERIMENTAL TECHNIQUES

To analyze receive side implementation we:

- Created an preliminary implementation library that uses existing MPI calls
- Extended the benchmarks from the Finepoints paper to measure extra recv compute time
- Ran experiments to evaluate overhead and benefit

4. IMPLEMENTATION

Our test implementation is designed to match the semantics of the Partitioned Communication proposal in the MPI standard. Notable differences from the original Finepoints implementation includes; interface support for receive side partitioning, a new completion mechanic based on MPI's RMA that allows for partition completions to a single, an internal method of grouping completions to support a mismatching partition counts.

5. BENCHMARKS

To evaluate this we extended the Finepoints microbenchmark. This benchmark uses an equal number of receive side partitions as send side. Additionally it reports a new metric; extra receive side compute time. This metric is the time difference between the first receive side partition completing and the last.

6. EXPERIMENTS

All of the experiments in the paper leverage said benchmark with the underlying implementation. They were run on a Cray XC40 test bed (the same architecture as Cori and Trinity) on the Xeon E5-2698v3 partition. Each node has 2 sockets with 16 cores and 2 way SMT. The tests were run for 100 iterations, with at 0.1 second compute time and 4% noise (which we expect to be common for exascale applications). Sweeps were done over threads and message size. To measure the potential overhead of the changes we compared the original metric of perceived bandwidth. To measure the potential for performance improvement, we compared the extra receive side compute time with the perceived comm time.

7. EXPERIMENTAL RESULTS

7.1. Overhead

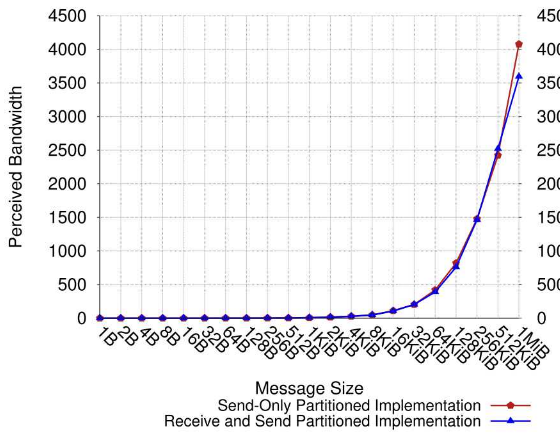
Figure 7-1 shows the overhead of the new implementation in terms of perceived bandwidth. In most cases both the original implementation and the new implementation have a similar bandwidth with a slight divergence at 1MiB messages. However, this changes at 32 threads and 64 threads. At 32 threads, a thread per core, the new implementation appears to have trouble handling large messages. At 64 threads, a thread per SMT, the new implementation appears to perform significantly better at message sizes >32KiB.

7.2. Extra Receive-Side Computation Time

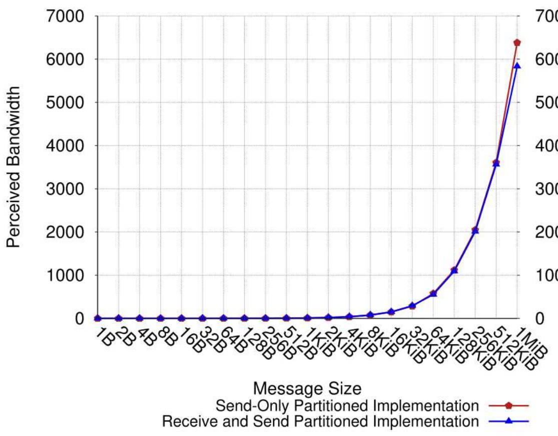
Figure 7-2 shows the extra compute time available to the processes through out the test (for all 100 iterations). At 1 thread there is no extra compute time; this makes sense as there is only a single RMA put, so there isn't a difference between a partition complete and a full operation completion. For the other thread counts it appears there is a roughly constant improvement to the available compute time, roughly equating to the computational noise. However, at 64 threads, the communication time seems to get elevated. This is likely due to the resource oversubscription associated with running a thread per SMT.

8. RELATED WORK

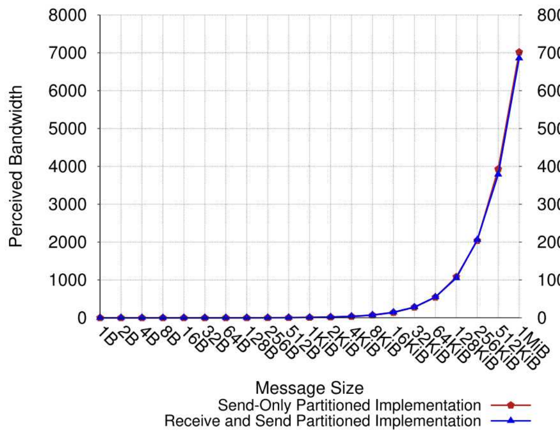
The goal of integrating thread/task programming models with MPI has been explored, with results showing that spreading communication traffic out in time can have performance



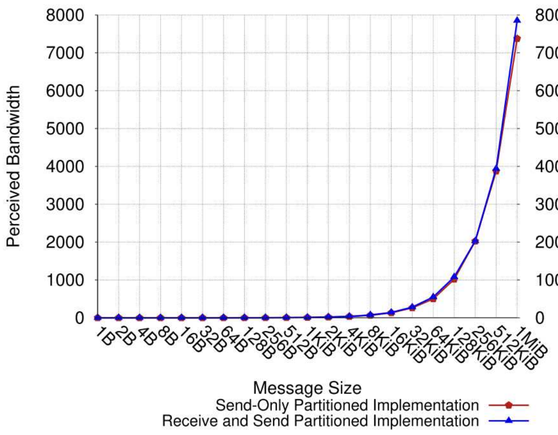
(a) 1 Thread



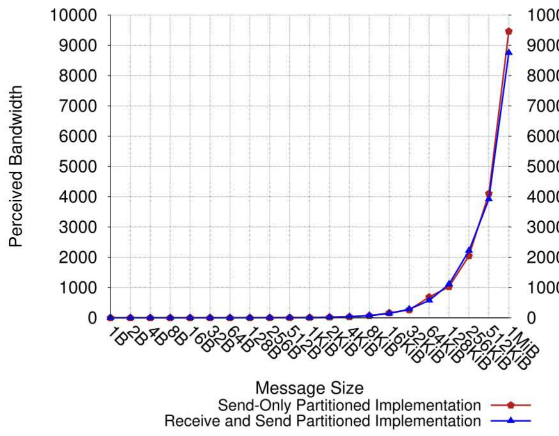
(b) 2 Threads



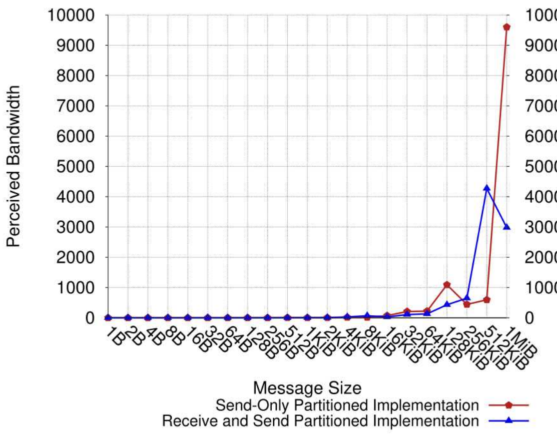
(c) 4 Threads



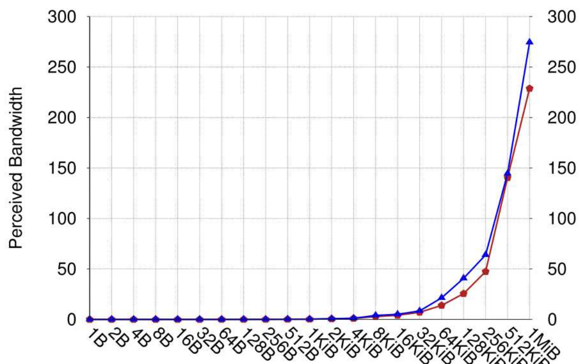
(d) 8 Threads

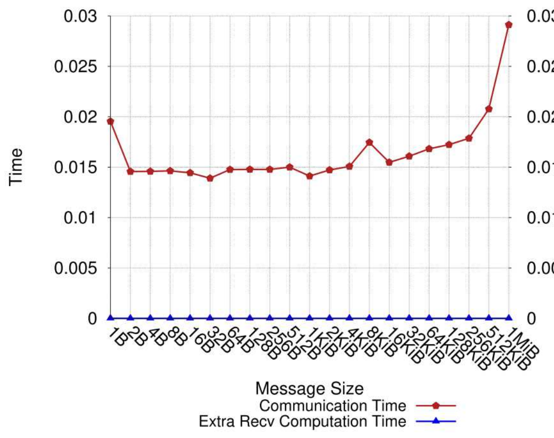


(e) 16 Threads

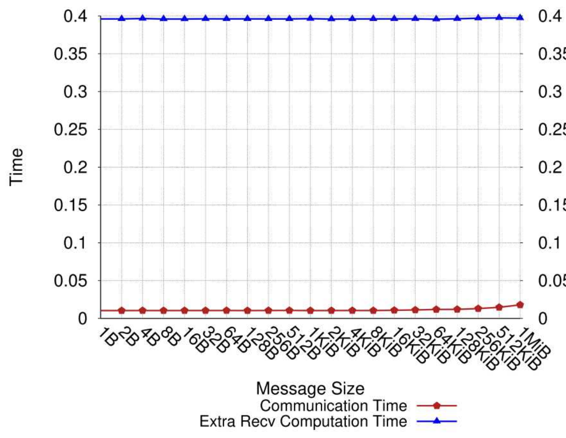


(f) 32 Threads

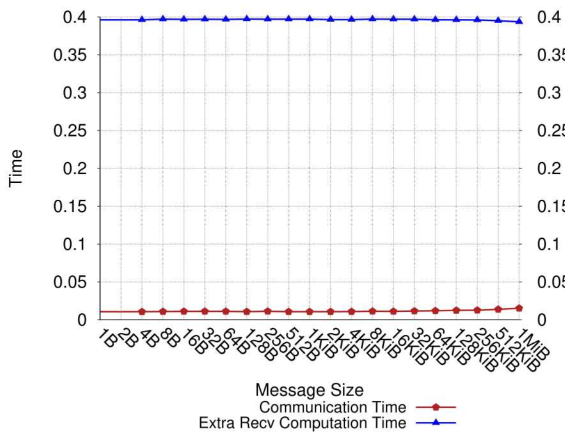




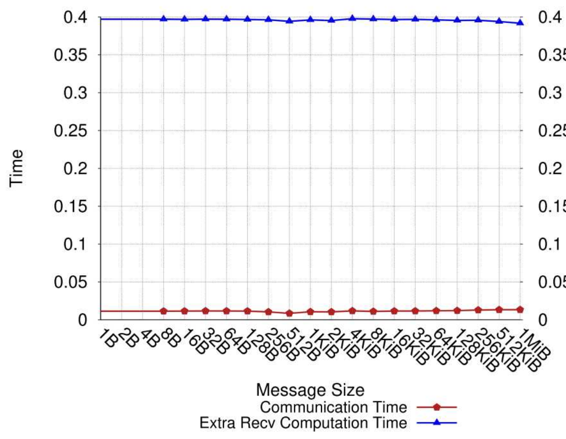
(a) 1 Thread



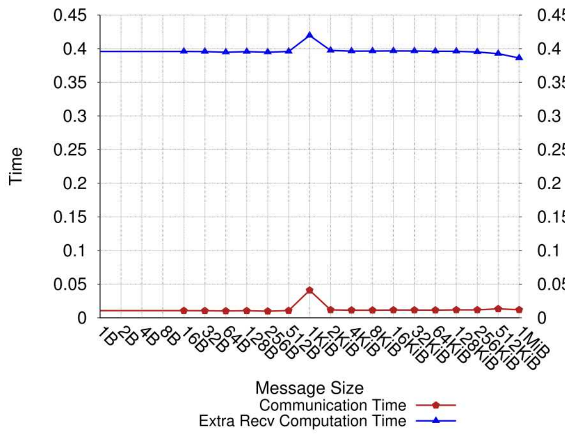
(b) 2 Threads



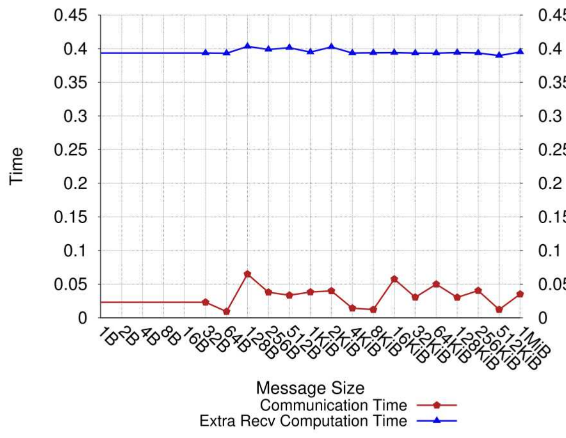
(c) 4 Threads



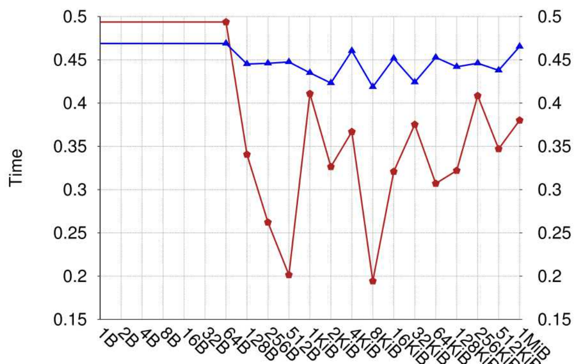
(d) 8 Threads



(e) 16 Threads



(f) 32 Threads



benefits [39, 4]. However, past work does not address new interfaces with MPI and relies on many smaller messages to enable performance.
Other approaches exist to minimize impact of message matching for many threads like hardware offloading [19] approaches such as Portals NICs [3] like BXI [13], InfiniBand ConnectX matching offloads [35]. Matching can also occur in proposed smartNIC solutions that perform line-speed hardware matching like sPIN [29] and INCA [38]. Message overheads have been studied across architectures [2] and network interfaces [15, 6].
Approaches for threading with MPI have also been explored. FG-MPI is a fine-grained MPI implementation that uses threads in place of processes and can scale to a large number of active threads [32]. Hybrid programming support has been added to major MPI implementations including MPICH [1] and Open MPI [28]. Alternatives for handling many threads at once by removing the ability to use wildcards have also been explored [10]. Methods of reducing threading overhead [8] and internal MPI thread placement for progress have been investigated [11, 12].
The concept of communication-computation overlap is a well known one, although in the current body of research typically only refers to overlapping MPI messages in their entirety, not in partitions. Work has been done on enabling overlap with applications [9, 30, 34]. Communication/computation overlap has also been studied directly [7, 31].
Our previous work on finepoints [24, 25] centered on the partitioned nature on the sending-side and adapting applications to use partitioned communication. This work differs from previous work as it is the first to address the impact of receive-side partitioned communication.

9. CONCLUSIONS

This work has shown that receive-side partitioning can be beneficial in some cases. Where overheads do not exceed the time delta caused by non-synchronous thread/task completion and where work can be completed without the full message buffer on the receive-side. The amount of computational time that is recoverable depends on the ability of the receive-side process to be able to progress with a portion of the message data and the extent of the time from the first message partition becoming available and whole message completion.

10. ACKNOWLEDGMENTS

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
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.

REFERENCES

[1] BALAJI, P., BUNTINAS, D., GOODELL, D., GROPP, W., AND THAKUR, R. Fine-grained multithreading support for hybrid threaded mpi programming. *The International Journal of High Performance Computing Applications 24*, 1 (2010), 49–57.

[2] BARRETT, B. W., BRIGHTWELL, R., GRANT, R., HAMMOND, S. D., AND HEMMERT, K. S. An evaluation of MPI message rate on hybrid-core processors. *International Journal of High Performance Computing Applications 28*, 4 (2014), 415–424.

[3] BARRETT, B. W., BRIGHTWELL, R., GRANT, R. E., HEMMERT, S., PEDRETTI, K., WHEELER, K., UNDERWOOD, K., RIESEN, R., MACCABE, A. B., AND HUDSON, T. The Portals 4.1 networking programming interface. Tech. Rep. SAND2017-3825, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2017.

[4] BARRETT, R. F., STARK, D. T., VAUGHAN, C. T., GRANT, R. E., OLIVIER, S. L., AND PEDRETTI, K. T. Toward an evolutionary task parallel integrated MPI+X programming model. In *6th Intl. Workshop on Programming Models and Applications for Multicores and Manycores* (2015), ACM, pp. 30–39.

[5] BERNHOLDT, D. E., BOEHM, S., BOSILCA, G., VENKATA, M., GRANT, R. E., NAUGHTON, T., PRITCHARD, H., AND VALLEE, G. A survey of MPI usage in the U.S. Exascale Computing Project. *Concurrency and Computation: Practice and Experience* (2018). DOI: 10.1002/cpe.4851.

[6] BRIGHTWELL, R., RIESEN, R., AND UNDERWOOD, K. D. Analyzing the impact of overlap, offload, and independent progress for message passing interface applications. *The International Journal of High Performance Computing Applications 19*, 2 (2005), 103–117.

[7] BRIGHTWELL, R., AND UNDERWOOD, K. D. An analysis of the impact of mpi overlap and independent progress. In *Proceedings of the 18th annual international conference on Supercomputing* (2004), ACM, pp. 298–305.

[8] CARRIBAULT, P., PÉRACHE, M., AND JOURDREN, H. Enabling low-overhead hybrid mpi/openmp parallelism with mpc. In *International Workshop on OpenMP* (2010), Springer, pp. 1–14.

[9] DANALIS, A., KIM, K.-Y., POLLOCK, L., AND SWANY, M. Transformations to parallel codes for communication-computation overlap. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing* (2005), IEEE Computer Society, p. 58.

[10] DANG, H.-V., SNIR, M., AND GROPP, W. Towards millions of communicating threads. In *Proceedings of the 23rd European MPI Users' Group Meeting* (2016), ACM, pp. 1–14.

[11] DENIS, A., JAEGER, J., JEANNOT, E., PÉRACHE, M., AND TABOADA, H. Dynamic placement of progress thread for overlapping mpi non-blocking collectives on manycore processor. In *European Conference on Parallel Processing* (2018), Springer, pp. 616–627.

[12] Denis, A., Jaeger, J., and Taboada, H. Progress thread placement for overlapping mpi non-blocking collectives using simultaneous multi-threading. In *European Conference on Parallel Processing* (2018), Springer, pp. 123–133.

[13] Derradji, S., Palfer-Sollier, T., Panziera, J.-P., Poudes, A., and Wellenreiter, F. The BXI interconnect architecture. In *Proceedings of the 23rd Annual Symposium on High Performance Interconnects* (2015), HOTI '15, IEEE.

[14] Dinan, J., Grant, R. E., Balaji, P., Goodell, D., Miller, D., Snir, M., and Thakur, R. Enabling communication concurrency through flexible MPI endpoints. *Int. Jour. of High Performance Computing Applications 28*, 4 (2014), 390–405.

[15] Doerfler, D., and Brightwell, R. Measuring mpi send and receive overhead and application availability in high performance network interfaces. In *European Parallel Virtual Machine/Message Passing Interface Users' Group Meeting* (2006), Springer, pp. 331–338.

[16] Dosanjh, M. G., Grant, R. E., Schonbein, W., and Bridges, P. G. Tail queues: A multi-threaded matching architecture. *Concurrency and Computation: Practice and Experience*, e5158.

[17] Dosanjh, M. G., Schonbein, W., Grant, R. E., Bridges, P. G., Gazimirsaeed, S. M., and Afsahi, A. Fuzzy matching: Hardware accelerated mpi communication middleware. In *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)* (2019), pp. 210–220.

[18] Dosanjh, M. G. F., Groves, T., Grant, R. E., Brightwell, R., and Bridges, P. G. RMA-MT: a benchmark suite for assessing MPI multi-threaded RMA performance. In *16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)* (2016), IEEE, pp. 550–559.

[19] Ferreira, K., Grant, R. E., Levenhagen, M. J., Levy, S., and Groves, T. Hardware mpi message matching: Insights into mpi matching behavior to inform design. *Concurrency and Computation: Practice and Experience*, e5150.

[20] Ferreira, K. B., Levy, S., Pedretti, K., and Grant, R. E. Characterizing MPI matching via trace-based simulation. In *Proceedings of the 24th European MPI Users' Group Meeting* (New York, NY, USA, 2017), EuroMPI '17, ACM, pp. 8:1–8:11.

[21] Ferreira, K. B., Levy, S., Pedretti, K., and Grant, R. E. Characterizing mpi matching via trace-based simulation. *Parallel Computing 77* (2018), 57–83.

[22] Flajslik, M., Dinan, J., and Underwood, K. D. Mitigating MPI message matching misery. In *International Conference on High Performance Computing* (2016), Springer, pp. 281–299.

[23] Ghazimirsaeed, S. M., Grant, R. E., and Afsahi, A. A dynamic, unified design for dedicated message matching engines for collective and point-to-point communications. *Parallel Computing 89* (2019), 102547.

[24] GRANT, R., SKJELLUM, A., AND BANGALORE, P. V. Lightweight threading with MPI using persistent communications semantics. Tech. rep., Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States), 2015.

[25] GRANT, R. E., DOSANJH, M. G., LEVENHAGEN, M. J., BRIGHTWELL, R., AND SKJELLUM, A. Finepoints: Partitioned multithreaded mpi communication. In *International Conference on High Performance Computing* (2019), Springer, pp. 330–350.

[26] GROVES, T., GRANT, R. E., AND ARNOLD, D. NiMC: Characterizing and eliminating network-induced memory contention. In *IEEE Intl. Parallel and Distributed Processing Symposium (IPDPS)* (2016), IEEE, pp. 253–262.

[27] GROVES, T. L., GRANT, R. E., GONZALES, A., AND ARNOLD, D. Unraveling network-induced memory contention: Deeper insights with machine learning. *IEEE Transactions on Parallel and Distributed Systems 29*, 8 (2017), 1907–1922.

[28] HJELM, N., DOSANJH, M. G. F., GRANT, R. E., GROVES, T., BRIDGES, P., AND ARNOLD, D. Improving MPI multi-threaded RMA communication performance. In *Proc. of the Int. Conf. on Parallel Processing* (2018), pp. 1–10.

[29] HOEFLER, T., DI GIROLAMO, S., TARANOV, K., GRANT, R. E., AND BRIGHTWELL, R. spin: High-performance streaming processing in the network. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2017), ACM, p. 59.

[30] JACOBSEN, D., THIBAULT, J., AND SENOCAK, I. An mpi-cuda implementation for massively parallel incompressible flow computations on multi-gpu clusters. In *48th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition* (2010), p. 522.

[31] KAISER, T. H., AND BADEN, S. B. Overlapping communication and computation with openmp and mpi. *Scientific Programming 9*, 2, 3 (2001), 73–81.

[32] KAMAL, H., AND WAGNER, A. An integrated fine-grain runtime system for MPI. *Computing 96*, 4 (2014), 293–309.

[33] LEVY, S., FERREIRA, K. B., SCHONBEIN, W., GRANT, R. E., AND DOSANJH, M. G. Using simulation to examine the effect of mpi message matching costs on application performance. *Parallel Computing 84* (2019), 63–74.

[34] MARJANOVIC, V., LABARTA, J., AYGUADÉ, E., AND VALERO, M. Effective communication and computation overlap with hybrid mpi/smpss. In *ACM Sigplan Notices* (2010), vol. 45, ACM, pp. 337–338.

[35] MARTS, W. P., DOSANJH, M. G. F., SCHONBEIN, W., GRANT, R. E., AND BRIDGES, P. G. Mpi tag matching performance on connectx and arm. In *Proceedings of the 26th European MPI Users' Group Meeting* (New York, NY, USA, 2019), EuroMPI '19, ACM, pp. 13:1–13:10.

[36] MPI FORUM. MPI: A message-passing interface standard version 3.1. Tech. rep., University of Tennessee, Knoxville, 2015.

[37] SCHONBEIN, W., DOSANJH, M. G. F., GRANT, R. E., AND BRIDGES, P. G. Measuring multithreaded message matching misery. In *Proceedings of the International European Conference on Parallel and Distributed Computing* (2018).

[38] SCHONBEIN, W., GRANT, R. E., DOSANJH, M. G., AND ARNOLD, D. Inca: In-network compute assistance. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (2019), ACM.

[39] STARK, D. T., BARRETT, R. F., GRANT, R. E., OLIVIER, S. L., PEDRETTI, K. T., AND VAUGHAN, C. T. Early experiences co-scheduling work and communication tasks for hybrid MPI+X applications. In *Workshop on Exascale MPI* (2014), IEEE Press, pp. 9–19.

[40] THAKUR, R., AND GROPP, W. Test suite for evaluating performance of MPI implementations that support MPI_THREAD_MULTIPLE. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface*. Springer, 2007, pp. 46–55.

[41] ZOUNMEVO, J. A., AND AFSAHI, A. A fast and resource-conscious mpi message queue mechanism for large-scale jobs. *Future Generation Computer Systems 30* (2014), 265–290.

DISTRIBUTION

Hardcopy—External

Number of Copies	Name(s)	Company Name and Company Mailing Address

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	Ryan E. Grant	1423	1319
1	Matthew G.F. Dosanjh	1423	1319

Email—Internal [redacted]

Name	Org.	Sandia Email Address
Technical Library	01177	libref@sandia.gov