Towards Efficient Collectives for MPI Remote Memory Access

Alexey A. Paznikov
apaznikov@gmail.com

Abstract

In this paper, we present a broadcast algorithm ("one-to-all" communication scheme) designed to improve the efficiency of Remote Memory Access (RMA) model for parallel programming in distributed-memory HPC systems. The core idea behind RMA is that a process can directly read from or write to another process's memory without the latter's explicit involvement in message transmission. RMA has the potential to reduce overheads associated with synchronization by employing collective operations that allow a process to simultaneously access multiple segments of distributed data structures. Unfortunately, the current MPI standard offers users only a sequential linear interface, which exhibits limited efficiency. Our objective is to bridge this gap by proposing a more efficient algorithm and software implementation for collective operations within the RMA model. To achieve this, we address this issue by focusing on the data transmission scheme from one process to all processes, and we introduce an algorithm based on process arrangement into a binomial tree – a method widely employed in classical collective operations in the MPI standard.

Keywords: MPI, RMA, remote memory access, one-sided communications, collective communications, collectives, broadcast.

1 Introduction

High-performance computing systems (HPC) have traditionally been employed for processing vast volumes of data. The complexity of tasks and the scale of processed data continue to increase. For instance, the sizes of social graphs in popular social networks are measured in petabytes [1, 2], and the number of graph nodes processed in HPC Graph500 benchmarks surpasses several trillion [3]. The data volumes generated by the Large Hadron Collider (LHC) reach hundreds of petabytes, even after multi-level data filtering [4, 5]. Expectations include application data volumes surpassing the petabyte barrier, requiring advances in current computing capabilities [6–9]. Analyzing such datasets necessitates hierarchically large-scale distributed HPC systems, with data residing in computational nodes' memory and represented as shared distributed data structures.

To enhance the performance of distributed HPC systems, performance enhancements are attained through the use of multi-processor and multi-core architectures and highly efficient communication networks that support Remote Direct Memory Access (RDMA) technology. While most existing applications rely on the message-passing model, this approach may become inadequate for future tasks due to significant synchronization overhead and physical limitations, such as reduced memory per data processing element, shared caches, and TLB buffers.

As a result, alternative software models are actively researched currently, including Remote Memory Access (RMA), MPI+X (MPI+threads), and MPI+MPI (MPI with shared memory) [10–15]. Recent studies have shown that in many applications, these models offer higher performance compared to the message-passing model. One of the most promising parallel programming models is RMA, implemented as a subsystem of one-sided communications in the MPI standard.

One-sided communication procedures enable a process to perform read or write operations on another process's memory without that process's direct involvement. Many applications demonstrate acceleration when replacing message send and receive operations with direct remote memory access [10–12, 16, 17]. Programming with RMA employs various techniques. From a hardware perspective, most distributed HPC systems support RDMA technology, which frees inter-node communication from the involvement of the central processor and the operating system [10–12, 18–22]. RDMA is supported by various networks, such as Infiniband [21, 22], IBM PERCS [23], Cray Gemini [24], Cray Aries [25], Omni-Path [26], and RoCE over Ethernet [27], among others.

The RMA model is closely linked to the Partitioned Global Address Space (PGAS) model [28, 29], a programming model actively developed today. PGAS provides a transparent interface for operations on distributed data structures. RMA and PGAS methods are used in many parallel languages, models, and tools, including MPI [30–32], SHMEM [33], Unified Parallel C [34], Co-array Fortran [35], Cray Chapel [36], and IBM X10 [37]. Unlike PGAS, RMA offers a lower-level interface and flexible synchronization control capabilities.

Let us describe the RMA model in MPI. In programs using RMA, processes exchange data by directly accessing each other's memory through non-blocking one-sided access operations [10–12] (Fig. 1, 2). `MPI_Put` operation for writing to remote memory and `MPI_Get` operation for reading from remote memory exhibit lower

latency compared to the message-passing model. Additionally, there is a set of atomic operations that implement linearizable remote access (`MPI_Accumulate`, `MPI_Get_accumulate`, `MPI_Fetch_and_op`, `MPI_Compare_and_swap`, and others, corresponding to similar operations in the shared memory model). RMA also includes operations that ensure data consistency and operation ordering, such as `MPI_Win_flush`/`MPI_Win_flush_all`.

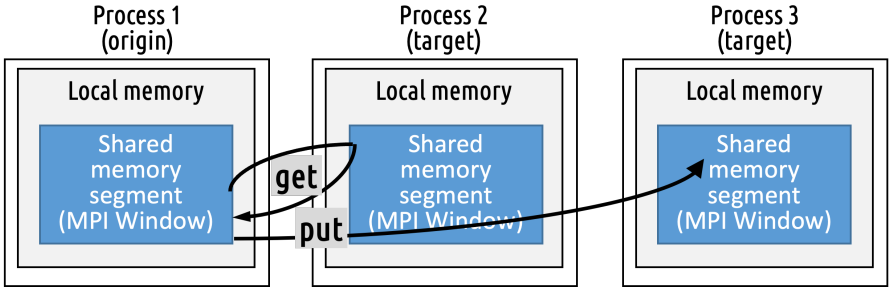

Figure 1: RMA window

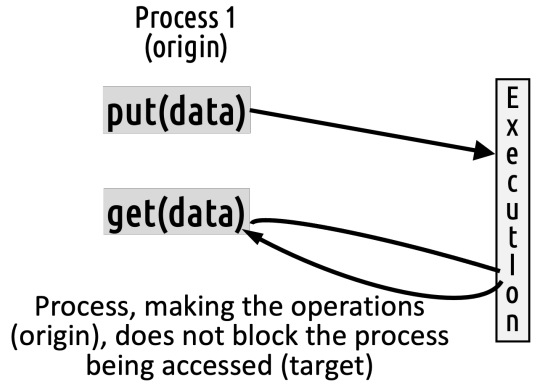

Figure 2: Non-blocking RMA-call principle

Within the framework of RMA, each process provides a portion of its local memory for remote access, referred to as a "window" (Fig. 1). These windows are shared among processes and can be accessed through RMA calls. All RMA operations take place within sections of code known as "epochs". Synchronization occurs within epochs, with two distinct types: "access epoch" and "exposure epoch." In an access epoch, only RMA calls intended for accessing the memory of remote (target) processes are allowed. Conversely, when a process is in an exposure epoch, other remote processes can perform RMA operations on the local memory of that process.

The model supports two types of synchronization: "active target" and "passive target." In active target synchronization, RMA calls are permitted only between two processes, one in an access epoch and the other in an exposure epoch. For implementing shared data structures, passive synchronization is appropriate, where only the accessing process participates. The concept is to allow access to target processes in a fully one-sided manner. Passive target synchronization is managed solely by the accessing process, requiring no actions on the part of the target processes.

In many cases, programs may involve numerous homogeneous RMA operations on multiple segments of remote memory [10–12, 17–20]. These operations are similar to collective operations in the message-passing model. Such patterns are frequently encountered in the implementation of distributed data structures, such as BCL [38], Active Message Queue [39], Hcl [40], or more scalable structures based on non-blocking synchronization [41, 42], as well as data structures with relaxed semantics [43].

Traditional collective operations are one of the most crucial components of MPI. Analysis of parallel algorithms and communication schemes reveals that over 80% of the total communication time is spent on them [44, 45]. For a wide range of parallel programs, the time spent on collective operations is a critical factor affecting overall performance. Presently, there are numerous algorithms available for implementing collective operations in the message-passing model [46–49]. However, even implementations that consider various hierarchical levels of HPC systems [19,50–52] still resemble classical collective operations and lack a one-sided interface. Efficient implementations of some collective operations have been proposed for the PGAS model [53, 54]. However, they cannot be directly applied in the RMA model, and they do not utilize the low-level performance tuning capabilities of the MPI RMA model.

To implement semantically equivalent collective operations in RMA, users of the current MPI standard are limited to using simple linear (sequential) algorithms. This approach is less efficient when the program's semantics allow for multiple parallel data access operations.

Given the information above, there is an interest in developing more efficient algorithms for collective operations in the RMA model. It is expected that these algorithms will significantly outperform simple linear schemes, thereby reducing the execution time of MPI programs and energy consumption. Let's address this challenge by considering the example of one-to-all broadcast communication and proposing an algorithm based on a binomial tree, one of the most popular methods used in implementing "one-to-all" communications.

2 Broadcast algorithms

The choice of a broadcast operation is motivated by its simplicity on one hand and its frequent usage on the other. The broadcast operation implements a "one-to-all" scheme: before the operation, data reside in the memory of a single process (root), and after the operation, the data are copied to all processes within the MPI communicator.

In terms of the RMA model, all participating processes must utilize a single RMA window called *win* with a sufficient allocated memory space. Additionally, unique identifiers are introduced for each RMA window (window id, *wid*) with corresponding runtime system support. The input parameters for the broadcast operation include the buffer $buf[0 \dots m]$, the RMA window *win* for broadcasting, its identifier *wid*, and an offset within the window (*disp*) that determines the buffer's location in the root process.

2.1 Linear broadcast algorithm

First, let us describe the linear algorithm (Fig. 3, 4). Within the passive target synchronization epoch for all processes (lines 2 and 6), the data in the buffer, denoted as buf, is broadcasted to the memory of all processes with $r = 1, 2, \dots, p$, corresponding to the communicator and window *win*. The algorithm comprises the loop in which, for each rank r within the communicator, an `MPI_Put` operation is executed to remotely write the buffer to process r at the specified offset *disp*. Additional synchronization is unnecessary, as all operations are guaranteed to be completed after the synchronization epoch.

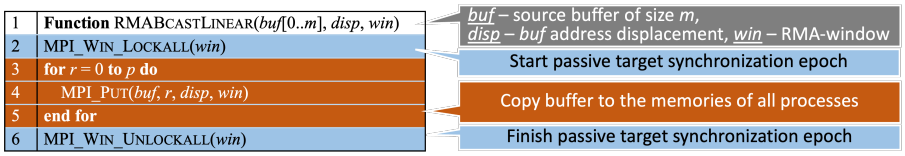

Figure 3: Linear broadcast algorithm in the RMA model: algorithm pseudocode

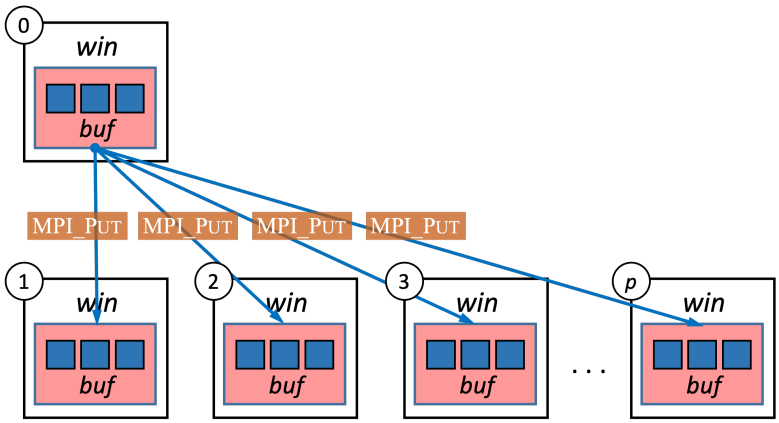

Figure 4: Linear broadcast algorithm in the RMA model: communication scheme

2.2 Binomial tree-based broadcast algorithm

In this work, we propose a broadcasting algorithm based on organizing processes into a binomial tree structure. Let's describe the runtime system for implementing this algorithm (Fig. 5). In addition to the *win* window

containing the broadcasted data, additional windows are created: *descrwin* holding information about the ongoing collective operation, *reqwin* corresponding to the request for operation execution, and *donecntrwin* containing a counter for processes that have completed the operation. Additionally, we create an auxiliary thread, the *Broadcaster* on each MPI process, which handles requests and facilitates further message relaying.

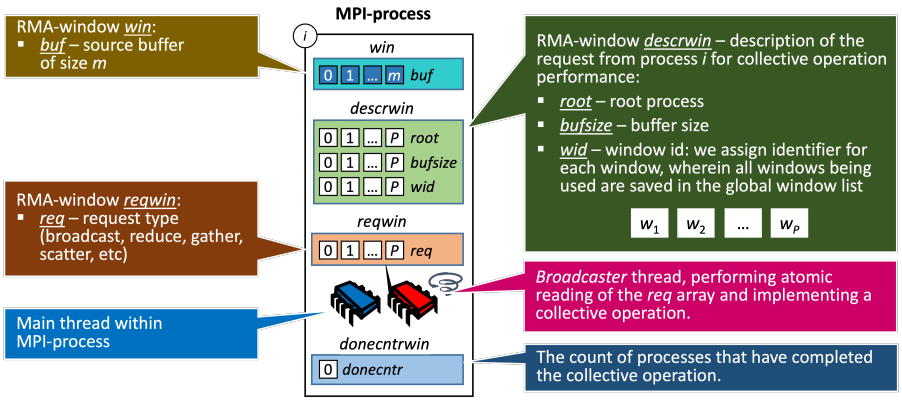

Figure 5: Runtime system for broadcast implementation in the RMA model

Before utilizing the RMA collective operation subsystem, the runtime system is initialized, during which the *Broadcaster* thread is launched on all processes. The main thread executes the `RMABcastBinomial` function, and its main steps are described (Fig. 6):

1. Wait for the completion of the previous operation (line 1) to prevent data mixing in the buffer during simultaneous broadcasts.
2. Initialize the request for executing the broadcast operation (line 2).
3. Wait for the *Broadcaster* thread to start and the synchronization epoch for additional windows to begin (line 3).
4. Call the `PutLoop` function to place data into the next processes according to the binomial tree structure (line 4).

The `PutLoop` function (Fig. 7) includes the following steps:

1. Calculate the shifted rank *srank*, which is the rank associated with the root process (line 2).
2. Execute a loop encompassing $\lceil log_2 p \rceil$ communication rounds (lines 5-16). In each round, each bit of the shifted rank *srank* is checked.
3. In each round, if the current bit in *srank* is not set (line 5), calculate the rank of the target process *prank* (line 9), and if a valid rank is obtained (line 7), perform the reverse translation from relative rank to the original rank (line 8). Then, send a request to the corresponding process in the `PutRequest` function (line 9). Otherwise, terminate the loop (line 12).

The `PutRequest` function implements copying the request, operation description, and data itself into the memory of the specified process *rank* (Fig. 8). Here, the previous process i can be any process that is not a leaf in the binomial tree, and j is any of its child elements. The main steps of the algorithm are as follows:

1. In the initial stage, the data buffer *buf* is copied to the remote process (line 1).
2. Next, an operation description *descr* is prepared, consisting of the root process *root* and the window identifier *wid* (line 2). This *descr* is then copied to the *descrwin* window at the computed displacement (line 3).
3. Atomically set a flag in the *reqwin* window (request) to indicate the readiness of data and description (line 4).
4. Wait for the completion of RMA calls, ensuring that the order of `MPI_Win_flush` functions guarantees the flag-setting operation occurs after copying data and the operation description (lines 5-7).

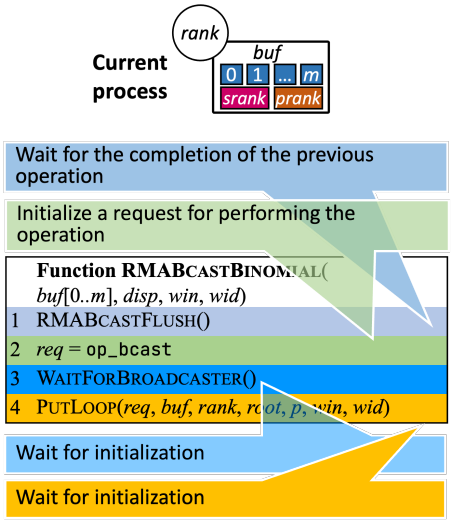

Figure 6: Binomial tree-based broadcast algorithm in the RMA model

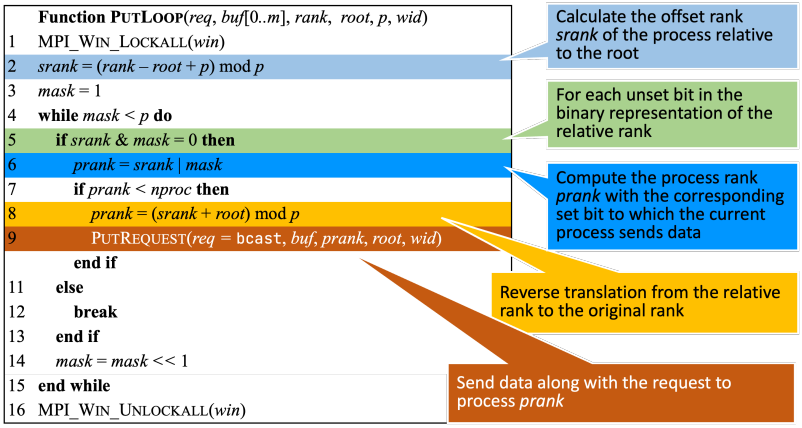

Figure 7: PutLoop function to put the data to other processes according binomial tree

Now, let's describe the operation of the *Broadcaster* thread, responsible for communication on all processes except the root (Fig. 9). This thread actively waits for incoming requests and data and then puts them further to other processes according the binomial tree. At the beginning of its execution, it opens epochs for the *reqwin* and *descrwin* windows and signals the start of the epoch. The main algorithm includes the following steps:

1. In an infinite loop (lines 1-15), atomically read the array of requests from local memory (lines 2-3) and find active requests (lines 4-13). After each iteration, wait for a timeout (line 14).

2. If a broadcast execution request from process i is found in the array of requests (line 5), it means that at that moment, process j already has the data buffer buf and the operation description $descr$ copied. Using the wid identifier, it locates the window corresponding to it (line 6), and calls `PutLoop` for further data broadcast.

3. After completing data transmission, atomically increment the counter in the root process (lines 8-10).

In addition to the broadcast operations themselves, synchronization functions need to be implemented. To address this, each process maintains a counter named *donecntr* (associated with the *donecntrwin* window) to keep track of the number of processes that have completed the operation.

We can define two synchronization functions, namely `RMABcastFlush` and `RMABcastTest` (see Fig. 10). The `RMABcastFlush` function is responsible for waiting until the broadcast operation is completed, while `RMABcastTest` checks whether the operation has been completed or not. Both of these functions utilize the `MPI_Fetch_and_op` operation with `NO_OP` argument (which corresponds to atomic variable reading) to atomically fetch the counter's state from local memory. If the count of completed processes reaches the total number of processes, the operation is considered completed.

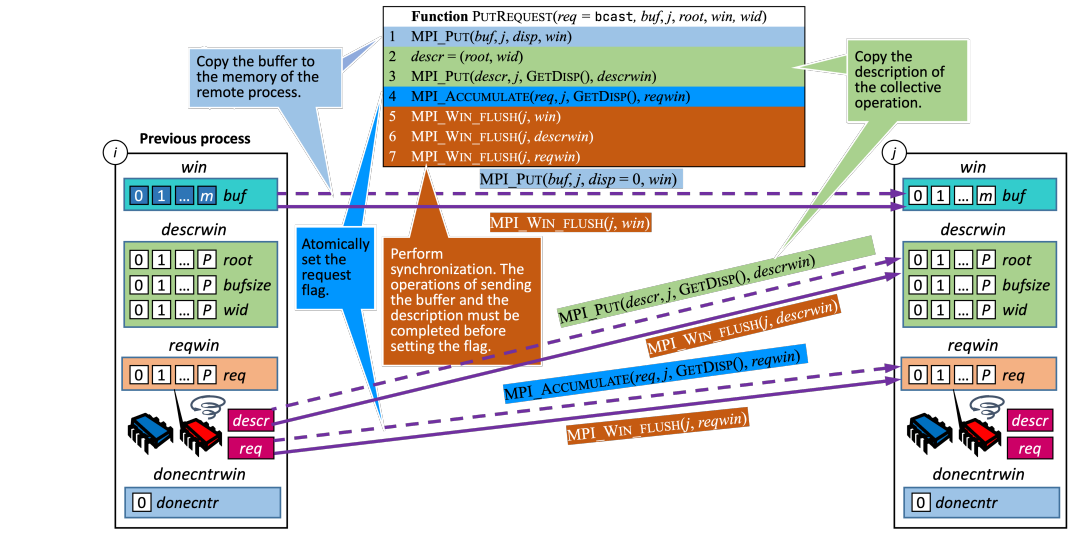

Figure 8: `PutRequest` execution scheme

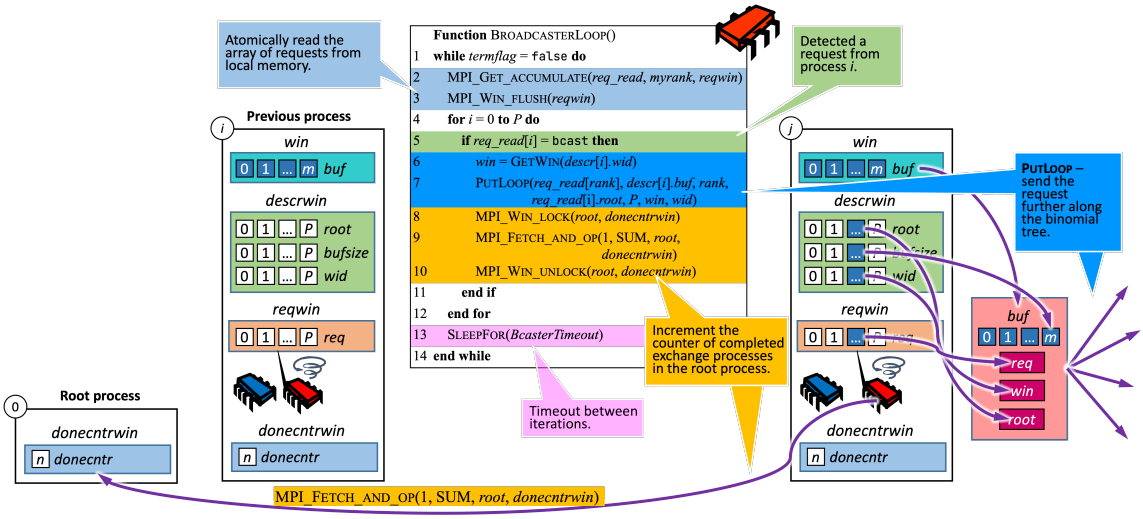

Figure 9: *Broadcaster* thread operation scheme

3 Analytical assessments in LogP and LogGP models

To assess the complexity of the RMA binomial-tree broadcast algorithm, we have developed analytical expressions for its execution time in the LogP and LogGP communication models. LogP serves as a communication model for distributed computing systems, where processes interact through short messages of fixed size, as described in [55]. The model's parameters include:

- L: Latency, representing the structural delays in message transmission between two processes.
- o: Overhead, accounting for the costs associated with organizing message sending or receiving.
- g: Minimum time gap between consecutive message sending or receiving operations.
- P: The total number of processes within the system.

Let's define o_r as the overhead associated with active waiting during request reception. The execution of the broadcast operation through a binomial tree can be divided into stages, the number of which corresponds to the height of the binomial tree for a power-of-two value of P: $\log_2 P$. The execution time for each stage is represented as $t_i = o + 2q + L + o_r$ (see Fig. 3).

Therefore, the overall execution time in the LogP model can be calculated as:

$$t = (o + 2q + L + o_r) \log_2 P + o_s + L$$

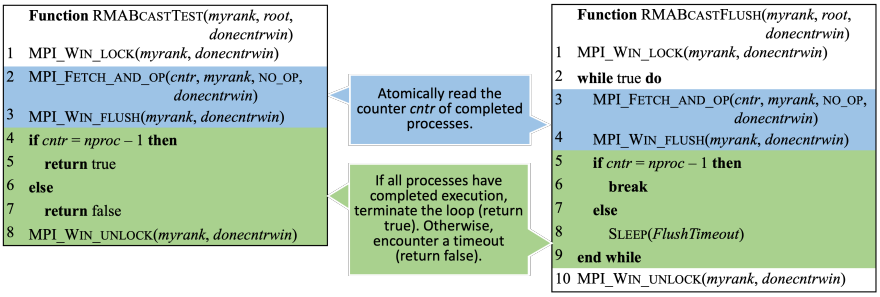

Figure 10: Synchronization functions for RMA broadcast

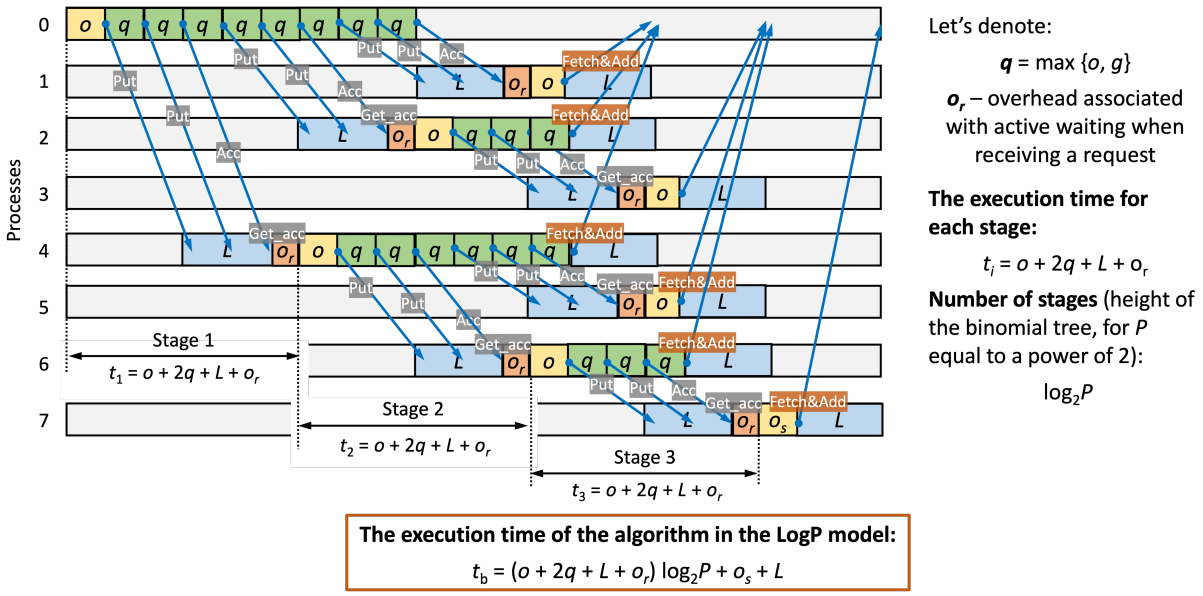

Figure 11: Binomial broadcast execution scheme in the LogP model

Binomial broadcast execution scheme in the LogP model Now, let's estimate the execution time of the algorithm in the LogGP model, which extends the LogP model to handle messages of larger sizes. In LogGP, the fifth parameter, G, represents the minimum time interval between consecutive sending or receiving of a single-byte-length message. Let's denote the message length as m. In this case, the execution time for each stage becomes $t_i = o + (m-1)G + 2q + L$ (Fig. 12).

The total execution time in the LogGP model for the operation in this scenario is given by:

$$t = (o + (m-1)G + 2q + L + o_r) \log_2 P + o + L$$

4 Experimental results

The experimental evaluation of broadcast algorithms was conducted using the clusters of the Center of Parallel Computational Technologies at the Siberian State University of Telecommunication and Information Sciences. Cluster 1 consists of 6 nodes, each equipped with 2 Intel Xeon E5620 processors, each having four cores (for a total of 48 cores). They are connected via Infiniband QDR. The library used is MVAPICH2 2.3.1. Cluster 2 comprises 18 nodes with 2 x Intel Xeon E5420 processors (4 processor cores each, totaling 144 cores). The communication network used is Gigabit Ethernet, and the MPI library employed is MPICH 3.2.1.

In the experiments, we designed a test that performed the RMA broadcast operation n times. An array of length $m = 2, 4, \ldots, 40$ MiB (for cluster 1) and $m = 2, 4, \ldots, 40$ KiB (for cluster 2) was used as the broadcast buffer. The number of processes, denoted as p, varied in the range of $8, 16, \ldots, 48$ (for cluster 1) and $8, 16, \ldots, 96$ (for cluster 2). The efficiency metric we employed is the average execution time, denoted as t.

Before the main measurements, a warm-up phase with n_w broadcast iterations was performed. Both $WaiterTimeout$ and $FlushTimeout$ parameters were set to 1 ms. The experiments were conducted for both the linear and binomial algorithms, with the root process set to 0 in all experiments.

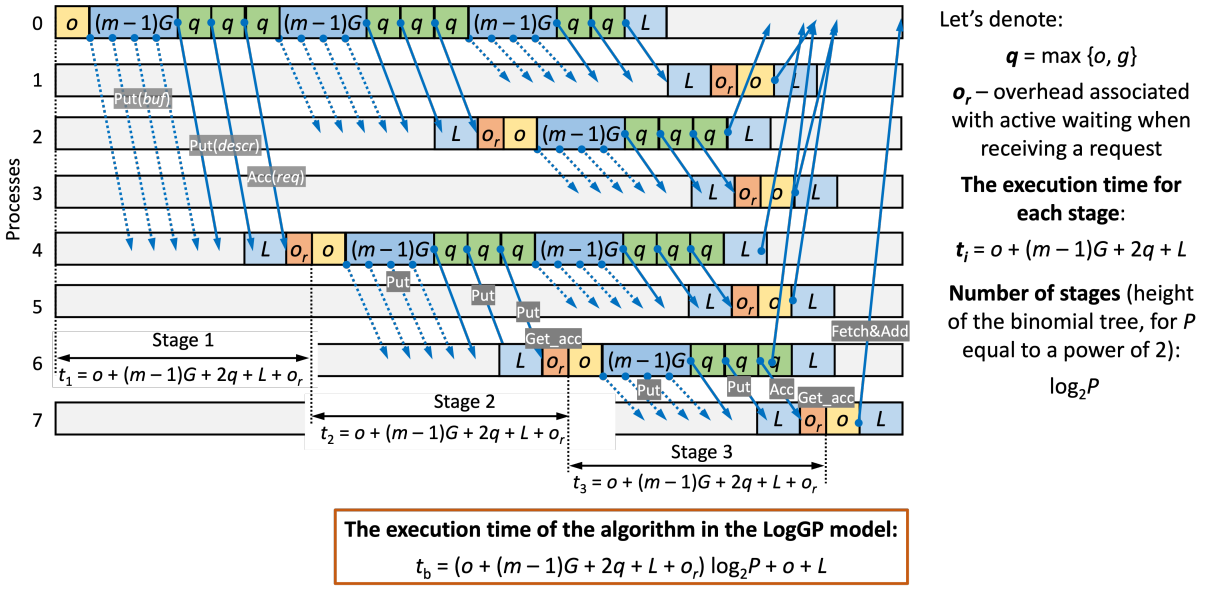

Figure 12: **Binomial broadcast execution scheme in LogGP model**

The experimental results are presented in Fig. 13, 14 (cluster 1), 15, 16 (cluster 2). Firstly, it's worth noting that the efficiency of the binomial algorithm begins to exceed that of the linear algorithm after a certain buffer size or number of processes. For example, in cluster 1 (Infiniband), with a buffer size of $m = 10$ MiB, this transition starts at $p = 32$ processes, and for $m = 16$ MiB, it begins at $p = 24$. In cluster 2 (Gigabit Ethernet), with a buffer size of $m = 16$ KB, the effect is observed at $p = 80$, and for $m = 34$ KB, it begins at $p = 32$. In general, the efficiency of the binomial algorithm increases with larger buffer sizes and more processes. This effect is more pronounced in lower-performance communication networks. However, for small buffers, the linear algorithm is more efficient due to the overhead costs of the binomial algorithm for process synchronization during execution. We believe that these overheads can be reduced by adjusting algorithm parameters. Additionally, a significant contribution to the total time comes from how the specific MPI library implements the `MPI_THREADS_MULTIPLE` mode, which is required for communication using two threads: the main and the *Broadcaster* threads.

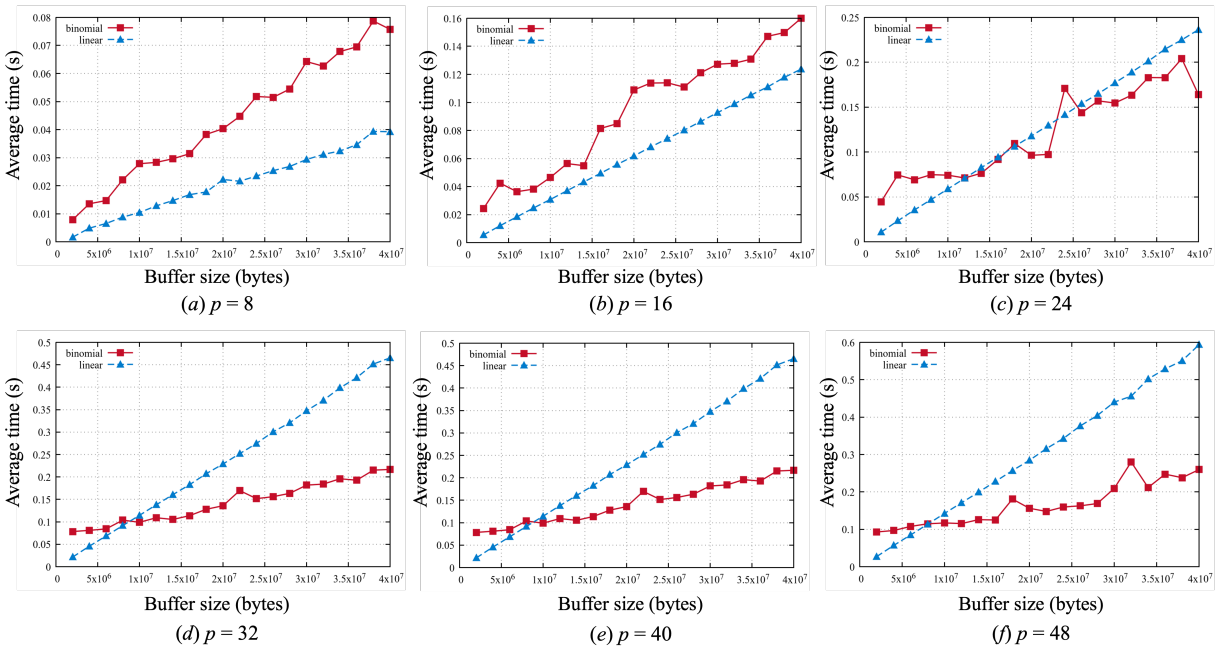

Figure 13: Comparison of latency for linear and binomial tree broadcast on buffer size (cluster 1, Infiniband QDR).

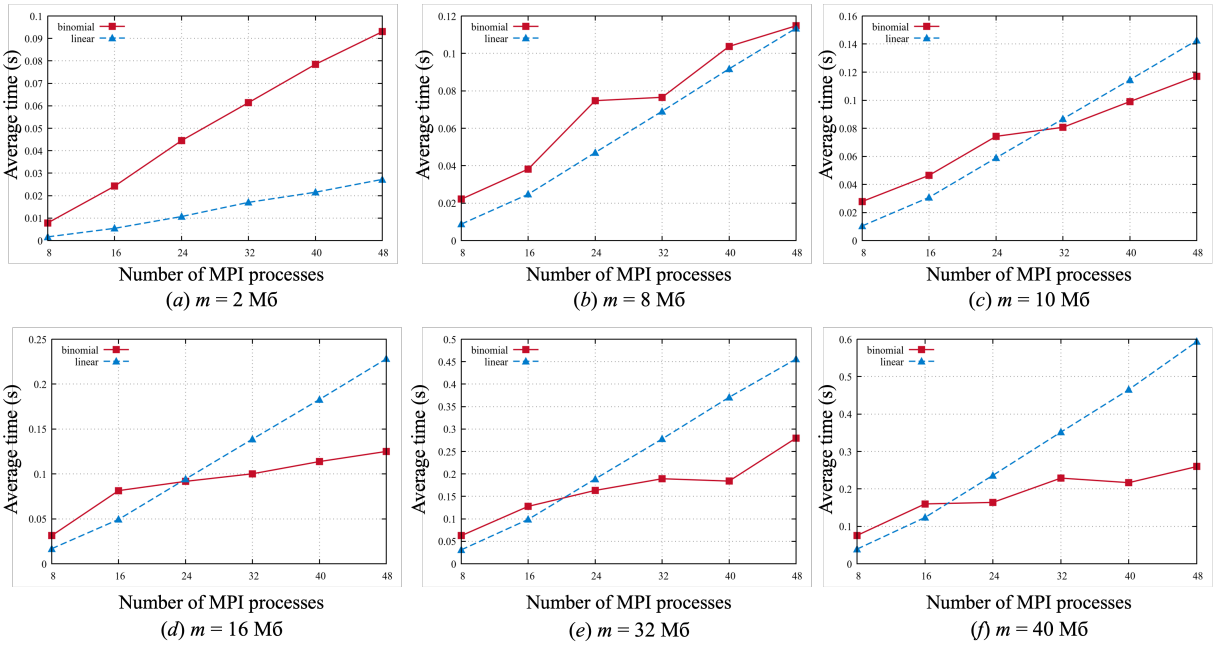

Figure 14: Comparison of latency for linear and binomial tree broadcast on process number (cluster 1, Infiniband QDR).

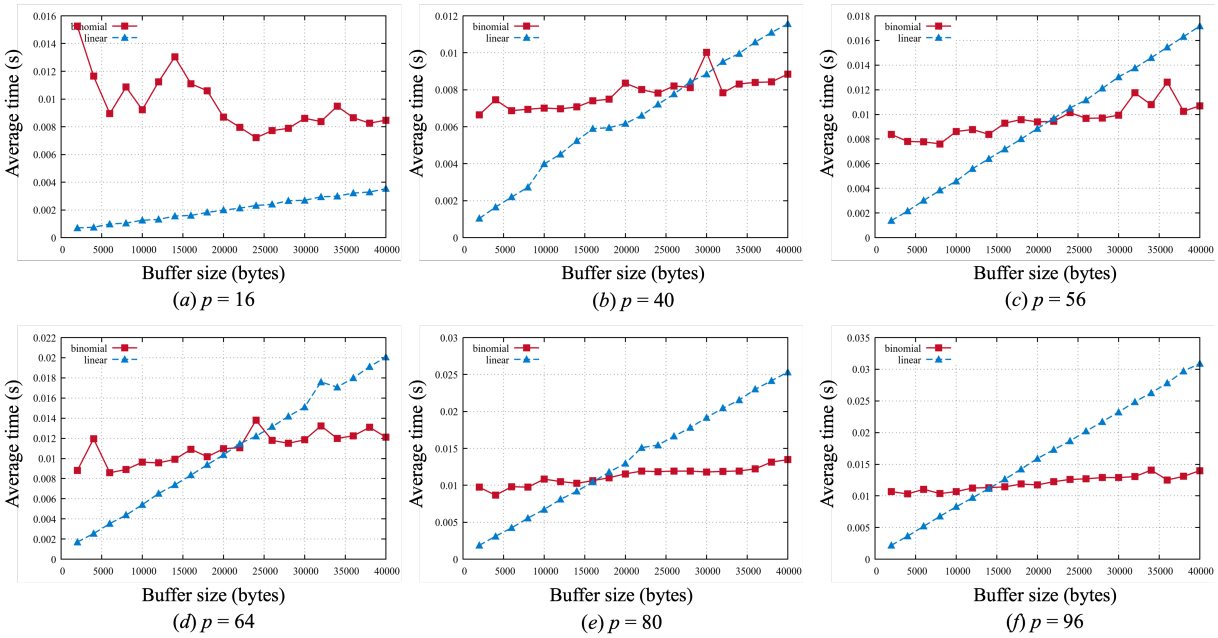

Figure 15: **Comparison of latency for linear and binomial tree broadcast on buffer size (cluster 2, Gigabit Ethernet).**

5 Conclusion

The current development and adoption of the RMA model in MPI represent the latest step in a trend that has emerged over the past decade, gradually synthesizing programming models for distributed and shared memory. RMA serves as a compromise between the classical message-passing model and the Partitioned Global Address Space (PGAS). It retains flexibility in managing operations and synchronization, allowing room for optimizations while also providing transparency for inheriting classical multi-threaded programming techniques.

However, there are several limitations in the current RMA standard that hinder its wider adoption. One prominent limitation is the absence of collective operations. To overcome this limitation, users are forced to resort to linear algorithms or transition to a message-passing model to utilize classical collective exchanges. The latter compromises program integrity and entails additional overhead.

This article aimed to explore the potential for overcoming this limitation. Using broadcast operations

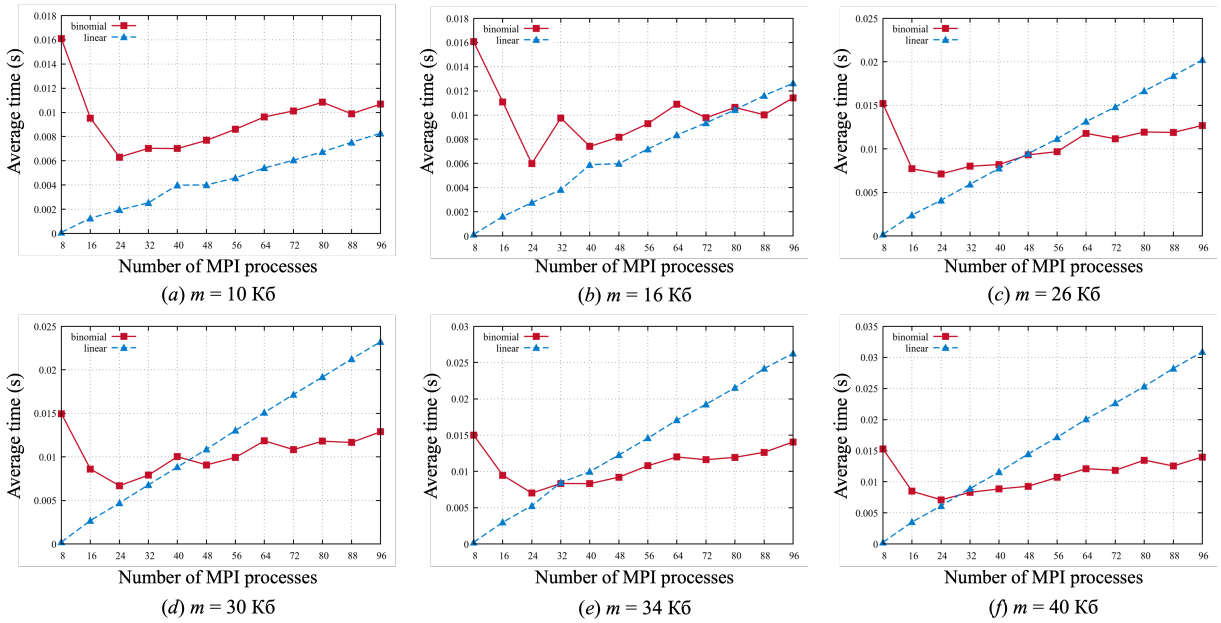

Figure 16: **Comparison of latency for linear and binomial tree broadcast on process number (cluster 2, Gigabit Ethernet).**

as an example, we demonstrated the possibility of achieving higher operational efficiency compared to the linear approach. The application of specialized broadcast implementation schemes, such as the binomial tree, significantly reduced the operation execution time. This effect becomes more pronounced with an increase in the total number of processes and the size of the transmitted buffer, making the new approach particularly valuable in large-scale systems and when processing large volumes of data. Further integration of collective RMA operations into MPI will undoubtedly require changes to the standard and substantial enhancements to library runtime systems, but, as we have shown, this is entirely feasible.

References

[1] N. Bronson, Z. Amsden, G. Cabrera, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, S. Kulkarni, H. Li, *et al.*, "{TAO}:{Facebook's} distributed data store for the social graph," in *2013 USENIX Annual Technical Conference (USENIX ATC 13)*, pp. 49–60, 2013.

[2] V. Venkataramani, Z. Amsden, N. Bronson, G. Cabrera III, P. Chakka, P. Dimov, H. Ding, J. Ferris, A. Giardullo, J. Hoon, *et al.*, "Tao: how facebook serves the social graph," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pp. 791–792, 2012.

[3] R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang, "Introducing the graph 500," *Cray Users Group (CUG)*, vol. 19, pp. 45–74, 2010.

[4] T. Åkesson, P. Eerola, V. Hedberg, G. Jarlskog, B. Lundberg, U. Mjörnmark, O. Smirnova, and S. Almehed, "Atlas computing: technical design report," *LHCC Reports; ATLAS Techical Design Reports*, 2005.

[5] G. Brumfiel *et al.*, "Down the petabyte highway," *Nature*, vol. 469, no. 20, pp. 282–283, 2011.

[6] C. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Information sciences*, vol. 275, pp. 314–347, 2014.

[7] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the future*, vol. 2007, no. 2012, pp. 1–16, 2012.

[8] K. M. Tolle, D. S. W. Tansley, and A. J. Hey, "The fourth paradigm: Data-intensive scientific discovery [point of view]," *Proceedings of the IEEE*, vol. 99, no. 8, pp. 1334–1337, 2011.

[9] C. Lynch, "How do your data grow?," *Nature*, vol. 455, no. 7209, pp. 28–29, 2008.

[10] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, and K. Underwood, "Remote memory access programming in mpi-3," *ACM Transactions on Parallel Computing (TOPC)*, vol. 2, no. 2, pp. 1–26, 2015.

[11] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling highly-scalable remote memory access programming with mpi-3 one sided," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–12, 2013.

[12] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda, "High performance rdma-based mpi implementation over infiniband," in *Proceedings of the 17th annual international conference on Supercomputing*, pp. 295–304, 2003.

[13] S. Sridharan, J. Dinan, and D. D. Kalamkar, "Enabling efficient multithreaded mpi communication through a library-based implementation of mpi endpoints," in *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 487–498, IEEE, 2014.

[14] T. Patinyasakdikul, D. Eberius, G. Bosilca, and N. Hjelm, "Give mpi threading a fair chance: A study of multithreaded mpi designs," in *2019 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 1–11, IEEE, 2019.

[15] T. Hoefler, J. Dinan, D. Buntinas, P. Balaji, B. Barrett, R. Brightwell, W. Gropp, V. Kale, and R. Thakur, "Mpi+ mpi: a new hybrid approach to parallel programming with mpi plus shared memory," *Computing*, vol. 95, pp. 1121–1136, 2013.

[16] P. Balaji, D. Buntinas, D. Goodell, W. Gropp, S. Kumar, E. Lusk, R. Thakur, and J. L. Träff, "Mpi on a million processors," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 16th European PVM/MPI Users' Group Meeting, Espoo, Finland, September 7-10, 2009. Proceedings 16*, pp. 20–30, Springer, 2009.

[17] J. Nieplocha, R. J. Harrison, and R. J. Littlefield, "Global arrays: A nonuniform memory access programming model for high-performance computers," *The Journal of Supercomputing*, vol. 10, pp. 169–189, 1996.

[18] D. Petrović, O. Shahmirzadi, T. Ropars, and A. Schiper, "High-performance rma-based broadcast on the intel scc," in *Proceedings of the twenty-fourth annual ACM symposium on Parallelism in algorithms and architectures*, pp. 121–130, 2012.

[19] V. Tipparaju, J. Nieplocha, and D. Panda, "Fast collective operations using shared and remote memory access protocols on clusters," in *Proceedings International Parallel and Distributed Processing Symposium*, pp. 10–pp, IEEE, 2003.

[20] S. Sur, U. K. R. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda, "High performance rdma based all-to-all broadcast for infiniband clusters," in *International Conference on High-Performance Computing*, pp. 148–157, Springer, 2005.

[21] G. F. Pfister, "An introduction to the infiniband architecture," *High performance mass storage and parallel I/O*, vol. 42, no. 617-632, p. 10, 2001.

[22] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen, "Design and implementation of mpich2 over infiniband with rdma support," in *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, p. 16, IEEE, 2004.

[23] B. Arimilli, R. Arimilli, V. Chung, S. Clark, W. Denzel, B. Drerup, T. Hoefler, J. Joyner, J. Lewis, J. Li, *et al.*, "The percs high-performance interconnect," in *2010 18th IEEE Symposium on High Performance Interconnects*, pp. 75–82, IEEE, 2010.

[24] R. Alverson, D. Roweth, and L. Kaplan, "The gemini system interconnect," in *2010 18th IEEE Symposium on High Performance Interconnects*, pp. 83–87, IEEE, 2010.

[25] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray cascade: a scalable hpc system based on a dragonfly network," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–9, IEEE, 2012.

[26] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, "Enabling scalable high-performance systems with the intel omni-path architecture," *IEEE Micro*, vol. 36, no. 4, pp. 38–47, 2016.

[27] M. Beck and M. Kagan, "Performance evaluation of the rdma over ethernet (roce) standard in enterprise data centers infrastructure," in *Proceedings of the 3rd Workshop on Data Center-Converged and Virtual Ethernet Switching*, pp. 9–15, 2011.

[28] K. Yelick, D. Bonachea, W.-Y. Chen, P. Colella, K. Datta, J. Duell, S. L. Graham, P. Hargrove, P. Hilfinger, P. Husbands, *et al.*, "Productivity and performance using partitioned global address space languages," in *Proceedings of the 2007 international workshop on Parallel symbolic computation*, pp. 24–32, 2007.

[29] C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, F. Cantonnet, T. El-Ghazawi, A. Mohanti, Y. Yao, and D. Chavarría-Miranda, "An evaluation of global address space languages: co-array fortran and unified parallel c," in *Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 36–47, 2005.

[30] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the mpi message passing interface standard," *Parallel computing*, vol. 22, no. 6, pp. 789–828, 1996.

[31] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, *et al.*, "Open mpi: Goals, concept, and design of a next generation mpi implementation," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 11th European PVM/MPI Users' Group Meeting Budapest, Hungary, September 19-22, 2004. Proceedings 11*, pp. 97–104, Springer, 2004.

[32] W. Gropp, R. Thakur, and E. Lusk, *Using MPI-2: Advanced features of the message passing interface*. MIT press, 1999.

[33] 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*, pp. 1–3, 2010.

[34] T. El-Ghazawi and L. Smith, "Upc: unified parallel c," in *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pp. 27–es, 2006.

[35] R. W. Numrich and J. Reid, "Co-array fortran for parallel programming," in *ACM Sigplan Fortran Forum*, vol. 17, pp. 1–31, ACM New York, NY, USA, 1998.

[36] B. L. Chamberlain, D. Callahan, and H. P. Zima, "Parallel programmability and the chapel language," *The International Journal of High Performance Computing Applications*, vol. 21, no. 3, pp. 291–312, 2007.

[37] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. Von Praun, and V. Sarkar, "X10: an object-oriented approach to non-uniform cluster computing," *Acm Sigplan Notices*, vol. 40, no. 10, pp. 519–538, 2005.

[38] B. Brock, A. Buluç, and K. Yelick, "Bcl: A cross-platform distributed data structures library," in *Proceedings of the 48th International Conference on Parallel Processing*, pp. 1–10, 2019.

[39] J. Schuchart, A. Bouteiller, and G. Bosilca, "Using mpi-3 rma for active messages," in *2019 IEEE/ACM Workshop on Exascale MPI (ExaMPI)*, pp. 47–56, IEEE, 2019.

[40] H. Devarajan, A. Kougkas, K. Bateman, and X.-H. Sun, "Hcl: Distributing parallel data structures in extreme scales," in *2020 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 248–258, IEEE, 2020.

[41] A. Burachenko, A. Paznikov, and D. Derzhavin, "Lock-free distributed queue in remote memory access model," *Tomsk State University Journal of Control and Computer Science*, no. 62, pp. 13–24, 2023.

[42] T.-D. Diep, P. H. Ha, and K. Fürlinger, "A general approach for supporting nonblocking data structures on distributed-memory systems," *Journal of Parallel and Distributed Computing*, vol. 173, pp. 48–60, 2023.

[43] A. Paznikov and A. Anenkov, "Implementation and analysis of distributed relaxed concurrent queues in remote memory access model," *Procedia Computer Science*, vol. 150, pp. 654–662, 2019.

[44] R. Rabenseifner, "Automatic mpi counter profiling," in *42nd cug conference*, pp. 396–405, 2000.

[45] D. Han and T. Jones, "Mpi profiling," tech. rep., Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2005.

[46] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of collective communication operations in mpich," *The International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[47] J. Pješivac-Grbović, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra, "Performance analysis of mpi collective operations," *Cluster Computing*, vol. 10, pp. 127–143, 2007.

[48] A. Paznikov and M. Kupriyanov, "Adaptive mpi collective operations based on evaluations in logp model," *Procedia Computer Science*, vol. 186, pp. 323–330, 2021.

[49] G. Almási, P. Heidelberger, C. J. Archer, X. Martorell, C. C. Erway, J. E. Moreira, B. Steinmacher-Burow, and Y. Zheng, "Optimization of mpi collective communication on bluegene/l systems," in *Proceedings of the 19th annual international conference on Supercomputing*, pp. 253–262, 2005.

[50] H. Zhu, D. Goodell, W. Gropp, and R. Thakur, "Hierarchical collectives in mpich2," in *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 16th European PVM/MPI Users' Group Meeting, Espoo, Finland, September 7-10, 2009. Proceedings 16*, pp. 325–326, Springer, 2009.

[51] S. Li, T. Hoefler, and M. Snir, "Numa-aware shared-memory collective communication for mpi," in *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pp. 85–96, 2013.

[52] J. L. Träff and A. Rougier, "Mpi collectives and datatypes for hierarchical all-to-all communication," in *Proceedings of the 21st European MPI Users' Group Meeting*, pp. 27–32, 2014.

[53] D. A. Mallón, G. L. Taboada, C. Teijeiro, J. González-Domínguez, A. Gómez, and B. Wibecan, "Scalable pgas collective operations in numa clusters," *Cluster computing*, vol. 17, pp. 1473–1495, 2014.

[54] I. Kulagin, A. Paznikov, and M. Kurnosov, "Heuristic algorithms for optimizing array operations in parallel pgas-programs," in *International Conference on Parallel Computing Technologies*, pp. 405–409, Springer, 2015.

[55] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. Von Eicken, "Logp: Towards a realistic model of parallel computation," in *Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*, pp. 1–12, 1993.