Design and Implementation of Key Proposed MPI-3 One-Sided Communication Semantics on InfiniBand

Sreeram Potluri, Sayantan Sur, Devendar Bureddy, and Dhabaleswar K. Panda

Department of Computer Science and Engineering, The Ohio State University
{potluri, surs, bureddy, panda}@cse.ohio-state.edu

Abstract. High-end computing systems have seen a tremendous growth in the recent years. Petaflop systems are already available and the researchers are actively working to address the challenges in achieving Exaflop performance. Programming models are a critical component in helping applications running on these machines achieve maximum performance. In order to achieve maximum performance, the programming model must enable simultaneous use of both processors and interconnection network. Over the years, MPI has become the standard for parallel application development. One-sided communication was introduced in MPI-2 to enable MPI developers to increase the utilization of system resource through overlap. However, studies have shown the limitations of the current model for both applications and higher-level libraries which would want to take advantage of the ubiquitous nature of MPI. As part of the MPI-3 effort, the Remote Memory Access group has proposed several extensions to the existing one-sided communication interface to address these limitations. In this paper, we present a design and implementation of some of the key semantics proposed for MPI-3 one-sided communication over InfiniBand. We have implemented our designs in the MVAPICH2 library. Experimental evaluation shows that our implementation of the dynamic windows provides similar communication performance as the existing implementation for static windows. The new flush semantics allow for better performance by reducing the local completion time of Put by 90% and of Get by 28% for an 8byte Message. The implementation of Get-Compute-Put benchmark using Request-based operations achieves 30% better performance compared to the one using only Lock-Unlock. It achieves optimal computation-communication overlap.

1 Introduction

High-end computing systems have seen a tremendous growth over the recent years. This has been driven by the increasing demand for compute cycles on one end and by the advances in processor, network and accelerator technologies on the other. As the capabilities of different components in a system increase, it is important for scientific applications to utilize all these components concurrently to achieve maximum performance. In particular, simultaneously utilizing the processors and interconnect is critical since network latencies are often the scaling bottlenecks. Programming models hold the key in enabling such usage. MPI had introduced non-blocking message passing and one sided communication semantics to address these requirements by enabling computation and communication overlap. Earlier work [7, 8] has shown how one sided communication semantics enable superior overlap in applications than the message passing semantics. However, adaptation of these semantics has been limited because of the overheads imposed by synchronization semantics of MPI-2 and a mismatch with real-world use cases for one-sided communication. Other one-sided models like Global

Address-Space Languages, and Global Arrays have failed to utilize the portable nature on MPI because of these limitations. As part of the MPI-3 effort, the Remote Memory Access (RMA) group has proposed several extensions to the existing model that promise to address many of these limitations.

Modern networks have played an indispensable role in scaling modern computing clusters. InfiniBand is a commodity interconnection network based on open standards. It has gained acceptance by the HEC community over the years. It is the primary interconnect in around 40% of the Top500 supercomputing clusters in the work. The Remote Direct Memory Access (RDMA) operations offered by InfiniBand free the processor from managing data transfers. This allows communication libraries to achieve higher performance and overlap.

1.1 Motivation

The proposed MPI-3 one-sided interface promises to address the limitations of the MPI-2 one-sided interface. The newer additions include dynamic window creation, light weight synchronization (local and remote) and variety of other communication operations. However, in order for wide spread acceptance of this proposed interface, the performance advantages of MPI-3 one-sided interface need to be clearly highlighted. We believe that this is a strong motivation for designing and implementing some of the key MPI-3 interfaces on a widely used commodity platform.

1.2 Contributions

This paper makes several important contributions. They are listed as follows:

1. To the best of our knowledge, this is the first design and implementation of key aspects of the proposed MPI-3 one-sided interface.
2. We present an analysis of the proposed MPI-3 extensions and through our experimental evaluation we establish that they solve several issues faced by the MPI-2 standard.
3. We present a new Get-Compute-Put benchmark using the new Request based operations.
4. Our design of the proposed semantics is integrated in the MVAPICH2 library [2], to demonstrate a working prototype in an open-source production MPI library.

The semantics implemented in this paper are marked in Figure 1. Experimental evaluation shows that our implementation of dynamic windows provides similar communication performance as the existing implementation for static windows despite the added flexibility. The new flush semantics allow for better performance by reducing the local completion time of a Put by 90% and of a Get by 28% for an 8byte Message. Our new Get-Compute-Put benchmark using Request-based operations achieves 30% better performance compared to the one using only Lock-Unlock. It achieves optimal computation-communication overlap.

2 Background

2.1 MPI-2 One-Sided Communication Semantics

The MPI-2 one-sided interface enables direct access to the memory of other processes through a "window". A window is a region in memory that each process exposes to other processes through a collective operation: MPI_Win_create. MPI-2 one-sided communication interface defines three types of data transfer operations. MPI_Put and MPI_Get transfer the data to and from a target window respectively. MPI_Accumulate combines the data movement to target with a reduce operation. All of these operations are non-blocking and are not guaranteed to complete,

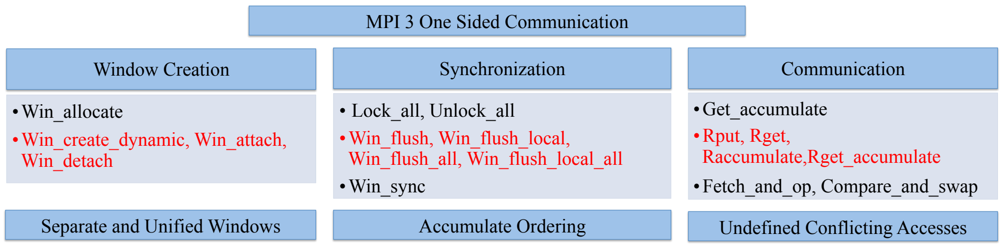

MPI 3 One Sided Communication
Window Creation
• Win_allocate
• Win_create_dynamic, Win_attach, Win_detach
Synchronization
• Lock_all, Unlock_all
• Win_flush, Win_flush_local, Win_flush_all, Win_flush_local_all
• Win_sync
Communication
• Get_accumulate
• Rput, Rget, Raccumulate,Rget_accumulate
• Fetch_and_op, Compare_and_swap
Separate and Unified Windows
Accumulate Ordering
Undefined Conflicting Accesses

Fig. 1. Proposed MPI-3 One-Sided Communication Standard Extensions

either locally or remotely, until a consequent synchronization operation. The period between two synchronization steps is termed as an epoch. Synchronization modes provided by MPI-2 can be classified into passive (no explicit participation from the target) and active (involves both origin and target). In the passive mode, an epoch is bounded by calls to MPI_Win_lock and MPI_Win_unlock and as the name suggests, this does not require any participation from the remote process.

2.2 Draft MPI-3 One-Sided Communication Semantics

The One-sided communication interface drafted by the MPI-3 RMA Working Group [1] retains all of the calls from MPI-2 while extending it in several ways. In this section, we only describe the calls that are the focus of this paper. Two new window creation mechanisms have been introduced: MPI_Win_allocate allocates the memory required for the window unlike MPI_Win_create which required the user to pass allocated memory; MPI_Win_create_dynamic allows users to dynamically attach and detach memory from a window through MPI_Win_attach and MPI_Win_detach calls. MPI_Rput, MPI_Rget, MPI_Raccumulate, MPI_Rget_accumulate behave like the corresponding regular communication operations but return request objects which enable the user to check for individual local completions. MPI_Win_flush_local and MPI_Win_flush calls have been introduced, which enable checking of bulk local and remote completion of communication operations to a target on a particular window without ending the access epoch. These calls can be used with passive synchronization only. MPI_Win_flush_all and MPI_Win_flush_local_all provide similar functionality but for a window at all the target processes.

The draft standard introduces a unified memory model to enable efficient implementations on cache-coherent systems. In this model, the target is guaranteed to get updated data from memory as soon as the operation initiated by an origin process has completed, without requiring any other MPI calls. The draft standard now enables ordering of overlapping accumulate operations in an epoch by default. All other operations are unordered. The user can relax the ordering by specifying appropriate info arguments during the window creation process. The new standard deems overlapping accesses as undefined rather than erroneous. This resolves compatibility issues with other programming models/languages which control the semantics of such accesses at a higher level (compiler/language).

3 Related Work

One-sided communication interface was originally introduced in MPI-2. Many libraries implement them over send/Receive Operations [10, 11]. Though these achieve reasonable performance, they cannot provide asynchronous progress and overlap. Several designs were proposed [6, 4, 9] to overcome these limitations by using RDMA-like features provided by modern interconnects. Though some applications have been modified to take advantage of the one-sided communication semantics [7, 8], their general adoption has been sparse. Researchers have pointed out the limitations of the one-sided semantics in MPI-2 [3, 12] for use by application developers and library writers. As part of the MPI-3 effort, the Remote Memory Access (RMA) group [1] has proposed extensions to the One-Sided Communication model to address many of these limitations. Our study in this paper focuses on providing an efficient implementation to some of the newly proposed MPI-3 one-sided semantics over InfiniBand.

4 Design and Implementation

In this work, we have considered a subset of the newly proposed features for MPI one-sided interface and have explored their designs and implementation on InfiniBand Clusters.

4.1 Dynamic Windows

Window creation is a collective operation. In MPI-2, the memory attached to a window is specified during window creation. It cannot be changed at a later point of time. We call this a "static" window. MPI-3 allows "dynamic" windows where each process can asynchronously attach or detach memory from a window.

The design of dynamic windows requires that the origin knows the address location of the buffer at the remote side. One option is to adopt a push model where we broadcast the registration information of any newly attached buffer at a process to all other processes in the communicator. However, this causes unnecessary messages on the network when only a small subset of processes access the window. This also does not scale with system size. The second design option is to use a rendezvous exchange when a process accesses a remote buffer for the first time. The origin process sends a Request-For-Info packet to the target. The target process responds with the registration information in a Response-With-Info packet. The origin process caches this information locally and initiates the communication transfer using RDMA. Later accesses to the buffer can be directly issued as RDMA operations. For small messages the request can be piggy-backed onto the actual data for the first message thus saving the round-trip time. The overhead of this initial exchange is easily amortized as buffers attached to a window are usually targets of multiple communication operations over the application's run-time. We follow this second design option in our implementation.

4.2 Flush Operations

All communication operations in MPI one-sided interface are non-blocking. In MPI-2, their completions are bound to synchronization operations. This is heavy-weight. MPI-3 addresses these issues through flush operations. Using flush operations, local completion can be distinguished from remote completion and does not require closing of access epochs. MPI_Win_flush_local and MPI_Win_flush_local_all ensure local completion of operations issued on a given window while MPI_Win_flush and MPI_Win_flush_all ensures remote completions.

Ensuring local and remote completions of different operations in InfiniBand have different requirements. Local and remote completion of RDMA reads and ATOMICS imply a corresponding event on the Completion Queue (CQ). This indicates that the operation completed both remotely and locally. However, CQ event for an RDMA write only means a local completion and does not ensure completion of the operation in the remote memory. The following excerpt from the InfiniBand specification [5] specifies the requirements to ensure remote completion. "An application shall not depend of the contents of an RDMA write buffer at the responder until one of the following has occurred: a) Arrival and Completion of the last RDMA write request packet when used with Immediate data, b) Arrival and completion of a subsequent send message, c) Update of a memory element by a subsequent ATOMIC operation".

Therefore, the remote completion of an MPI_Put requires extra handling when implemented over an RDMA Write. In our design, we issue a completion message which is acknowledged by the remote process. This ensures correct remote completion. For large remote memory accesses, we use an optimization by utilizing InfiniBand ATOMIC operations. Using ATOMIC operations, we can *asynchronously* provide remote memory completion.

The implementation of MPI_Win_flush_local (MPI_Win_flush_local_all) and MPI_Win_flush (MPI_Win_flush_all) depends on the communication operations issued on the window and how these operations are executed in the library. When only RDMA reads are issued, both flush_local and flush require to block for the corresponding CQ events only. When there are only RDMA writes, flush_local will require to block for CQ events while flush will have to wait for response to an extra send operation issued after the writes. When the communication operations are implemented as send operations by the library, flush_local requires to block for CQ events, while flush blocks for a response from the remote process, on the last send operation that was issued.

4.3 Request-based Operations

Request-based operations provide an easy mechanism to wait for completion of *certain* operations. This makes it much more finer grained than flush_local, which waits for local completion of all operations. We have shown sample pseudo-code for using this request-based operations in Figure 2. In this example, N blocks of data are fetched from remote memory, computed on, and written back to remote memory. The programmer should be able to pipeline the three phases: get, compute and put, completely hiding the communication cost using the new request based interface, as shown in Figure 2(c).

As discussed in the previous section 4.2, local completion in InfiniBand can be easily detected by the corresponding event on the completion queue. For RDMA read, a local completion means that the data is available in the local buffer and is ready to be used by the upper layer. This satisfies the request completion semantics of an MPI_Rget. For MPI_Rput, a local completion means that the user send buffer is free to be reused. Whether it is implemented over RDMA write or send, these requirements are satisfied by a local completion event on InfiniBand.

5 Experimental Results

In this section, we provide a detailed performance analysis of our designs. The tests were carried out on Intel Westmere platform. Each node has 8 cores running at 2.67GHz and is equipped with two Mellanox QDR MT26428 HCAs. Each node has 12GB of DDR3 RAM. The operating system used is Red Hat Enterprise Linux Server release 5.4. We have integrated and evaluated our designs with MVAPICH2 1.6 version.

5.1 Performance on Dynamic Windows

In this section, we compare the performance of MPI_Put and MPI_Get operations on the static and dynamic windows. Results in Figures 3(a) and 4(a) show the small message latency of Put and Get operations, respectively. We observe similar performance for both static and dynamic windows. The overhead of exchanging the registration information during the first communication call on the dynamic windows is clearly amortized. Large messages follow a similar

```
MPI_Win_lock
for i in 1, N
    MPI_Get (ith Block)
end for
MPI_Win_unlock

Compute (N Blocks)

MPI_Win_lock
for i in 1, N
    MPI_Put (ith Block)
end for
MPI_Win_unlock
```

(a) No Overlap

```
MPI_Win_lock
for i in 1, N
    MPI_Get (ith Block)
end for
MPI_Win_unlock

MPI_Win_lock
for i in 1, N
    Compute (ith Block)
    MPI_Put (ith Block)
end for
MPI_Win_unlock
```

(b) Overlap using Lock-Unlock

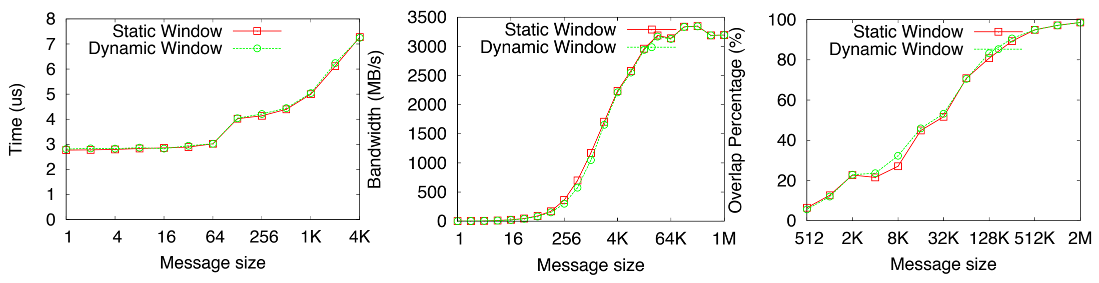
```
MPI_Win_lock
for i in 1, N
    MPI_Rget (ith Block)
end for

MPI_Wait_any (get requests)
while a get request j completes
    Compute (jth Block)
    MPI_Rput (jth Block)
    MPI_Wait_any (get requests)
end while
MPI_Wait_all (put requests)
MPI_Win_unlock
```

(c) Overlap using Request Operations

Fig. 2. Get-Compute-Put on N Blocks of Data

trend and have not been represented in the graphs. A comparison of bandwidth performance is shown in in Figures 3(b) and 4(b). As operations after the first exchange can be directly issued to the network as RDMA operations, we see similar overlap for operations on dynamic windows as on static windows. We can achieve more than 80% overlap for message sizes greater that 128Kbytes. Figures 3(c) and 4(c) show these results.

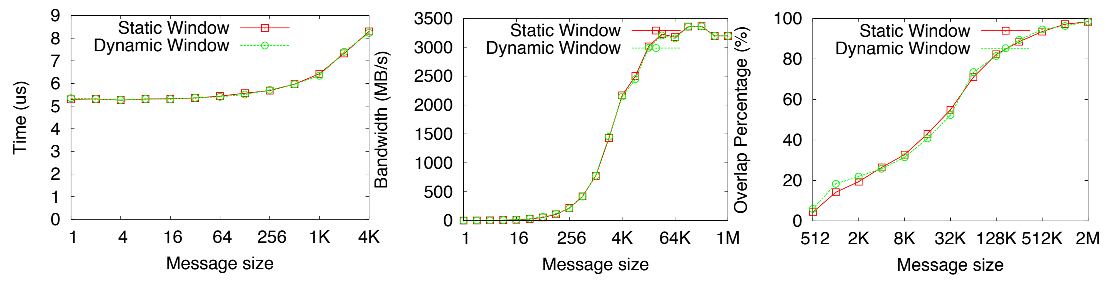


(a) Latency (b) Bandwidth (c) Overlap

Fig. 3. Put Performance on Dynamic Windows

5.2 Performance of Flush Operations

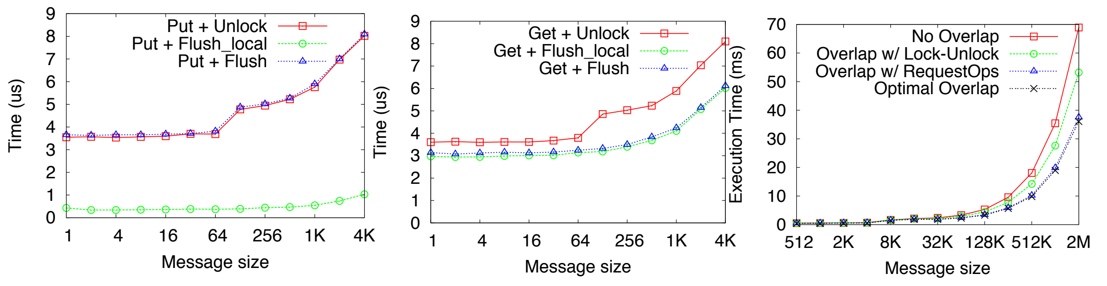
The results presented in this section show how the newly introduced flush operations can enable applications to check for completions more efficiently. Figure 5(a) compares the time for an MPI_Put and its completion using MPI_Win_Unlock, MPI_Win_Flush_local and MPI_Win_Flush. We see that a check for local completion, using MPI_Win_Flush_local, which blocks on a CQ event, takes only 0.34 usec for a 4byte message, while a check for remote completion using MPI_Win_Flush/MPI_Win_Unlock takes 3.6 usec. We block for a response from the remote process in the case of remote completion. Hence the new semantics allow for a 90large messages, we observed that local and remote completions take similar time.



(a) Latency (b) Bandwidth (c) Overlap

Fig. 4. Get Performance on Dynamic Windows

Figure 5(b) compares the completion times for a MPI_Get operation. MPI_Get implemented over RDMA read requires to block for a local CQ event and this means both local and remote completions. However, when MPI_Win_unlock is used, MPI_Get is implemented as a send/receive exchange with unlock operation piggybacked onto it. Hence we see that completion of smaller messages using MPI_Win_Flush_local and MPI_Win_Flush take 28% less time compared to completion using MPI_Win_unlock. For larger messages, the transfer are implemented using RDMA and Unlock is sent as a separate message. The latencies for larger messages have been observed to be similar in all the cases. Hence, we have not include the corresponding graphs.



(a) Put (b) Get (c) Get-Compute-Put

Fig. 5. (a) Local and Remote Completion of Put using Flush Operations. (b) Local and Remote Completion of Get using Flush Operations (c) Overlap using Request Based Operations in a Get-Compute-Put Model

5.3 Overlap using Request-based Operations

Results in this section show how Request-Based Operations can be used to achieve superior overlap compared to MPI-2. We use the Get-Compute-Put pattern discussed in Section 4.3. "No Overlap" refers to 2(a), "Overlap w/ Lock-Unlock" refers to Figure 2(b) and "Overlap w/ RequestOps" refers to Figure 2(c). "Optimal Overlap" is the best theoretically possible overlap performance which is equal to the computation time. The number of blocks N used was 25 and with computation time set to greater than the communication time. Results in Figure 5(c) show that the code using Request-based Operations can achieve close to optimal overlap and performs close to 30% better than the version using only MPI_Win_lock and MPI_Win_unlock calls for 4 MB message size.

6 Conclusion and Future Work

MPI-2 had introduced one-sided interface to enable applications to leverage RDMA feature offered by modern interconnects. However, the MPI-2 one-sided interface was not widely adopted due to several limitations. The proposed MPI-3 one-sided communication interface resolves many of these limitations. In this paper, we presented a design and implementation of a key subset of newly proposed one-sided interfaces. Through micro-benchmark evaluation, we have shown that the newly proposed interfaces can provide improved performance over the MPI-2. In the near future, we would like to show these benefits using a real-world application, re-designing it to new the new functions and semantics.

7 Acknowledgments

This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CCF-0833169, #CCF-0916302, #OCI-0926691 and #CCF-0937842; grants from Intel, Mellanox, Cisco, QLogic, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Appro, Chelsio, Dell, Microway, QLogic, and Sun Microsystems.

References

1. MPI-3 RMA Working Group. `http://meetings.mpi-forum.org/mpi3.0_rma.php`
2. Network Based Computing Laboratory, MVAPICH2. http://mvapich.cse.ohio-state.edu/
3. Bonachea, D., Duell, J.: Problems with Using MPI 1.1 and 2.0 as Compilation Targets for Parallel Language Implementations. Int. J. High Perform. Comput. Netw. 1(1-3), 91–99 (2004)
4. Huang, W., Santhanaraman, G., Jin, H.W., Panda, D.K.: Scheduling of MPI-2 One Sided Operations over InfiniBand. In: IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 9. p. 215.1. IEEE Computer Society, Washington, DC, USA (2005)
5. InfiniBand Trade Association: InfiniBand Architecture Specification, Release 1.2 (October 2004)
6. Jiang, W., Liu, J., Jin, H.W., Panda, D.K., Gropp, W., Thakur, R.: High performance MPI-2 One-sided Communication over InfiniBand. In: CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid. pp. 531–538. IEEE Computer Society, Washington, DC, USA (2004)
7. Mirin, A.A., Sawyer, W.B.: A scalable implementation of a finite-volume dynamical core in the community atmosphere model. Int. J. High Perform. Comput. Appl. 19(3), 203–212 (2005)
8. Potluri, S., Lai, P., Tomko, K., Sur, S., Cui, Y., Tatineni, M., Schulz, K., Barth, W., Majumdar, A., Panda, D.K.: Quantifying Performance Benefits of Overlap using MPI-2 in a Seismic Modeling Application. In: International Conference on Supercomputing (ICS'10) (2010)
9. Santhanaraman, G., Balaji, P., Gopalakrishnan, K., Thakur, R., Gropp, W., Panda, D.K.: Natively Supporting True One-Sided Communication in MPI on Multi-core Systems with InfiniBand. In: CC-GRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 380–387. IEEE Computer Society, Washington, DC, USA (2009)
10. Thakur, R.: An evaluation of implementation options for mpi one-sided communication. In: In Recent Advances in Parallel Virtual Machine and Message Passing Interface, 12th European PVM/MPI Users Group Meeting. pp. 415–424. Springer (2005)
11. Thakur, R., Toonen, B.: Minimizing synchronization overhead in the implementation of mpi one-sided communication. In: Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users Group Meeting, pages 57 67. Lecture Notes in Computer Science 3241. pp. 57–67. Springer (2004)
12. V. Tipparaju and W. Gropp and H. Ritzdorf and R. Thakur and J. L. Träff: Investigating High Performance RMA Interfaces for the MPI-3 Standard. In: In Proceedings of the 2009 International Conference on Parallel Processing (ICPP) (2009)