Scheduling of MPI-2 One Sided Operations over InfiniBand [*]

Wei Huang Gopalakrishnan Santhanaraman Hyun-Wook Jin
Dhabaleswar K. Panda
Department of Computer Science and Engineering
The Ohio State University
Columbus, OH 43210
{*huanwei, santhana, jinhy, panda*}*@cse.ohio-state.edu*

Abstract

MPI-2 provides interfaces for one sided communication, which is becoming increasingly important in scientific applications. MPI-2 semantics provide the flexibility to reorder the one sided operations within an access epoch. Based on this flexibility, in this paper we try to improve the performance of one sided communication by scheduling one sided operations. We have come up with several re-ordering and aggregating schemes to achieve better network utilization. We have evaluated these schemes on both PCI-X and PCI-Express platforms. With re-ordering scheme, we see an improvement in the throughput up to 76%, latency up to 40%. With aggregation scheme, we observe an improvement of 44% and 42% for MPI_Put and MPI_Get latency respectively on PCI-Express platform.

1. Introduction

Scientific computing has seen a dramatic growth in the recent years. Parallel systems with increasingly large number of processors are being deployed as a means of attaining the computing power needed to sustain the development in this field. The advent of clusters of workstations powered by high performance networks have focused on distributed memory systems to support this need for higher computing power.

In the last decade MPI (Message Passing interface) [6] has evolved as the *de facto* parallel programming model for distributed memory systems. MPI has a number of features that provide both convenience and high performance communication. MPI-1 standard specifies message passing based on the send-receive model. Both the sender and the receiver are involved in the communication and the synchronization is achieved implicitly through the communication operation. This model is often referred to as the two sided communication model. As an extension to MPI-1, the MPI-2[12] standard introduces the one sided communication model also known as the Remote Memory Access (RMA) model. In this model ideally only one side is involved in the data communication process and the other side is unaware of it. Synchronization is done explicitly to ensure completion before using the data.

In the area of High performance networking, InfiniBand[7] has emerged as a strong player. InfiniBand architecture provides Remote Direct Memory Access (RDMA) capability with which we can directly access the remote address space. This fits in well with the RMA communication model.

Our research group has been actively involved in designing a high performance implementation of MPI-2 over InfiniBand[14]. Earlier we have proposed designs to implement two-sided communication by utilizing the RDMA features provided by InifiniBand[11]. We also proposed efficient and scalable designs for implementing one sided communication with both active and passive synchronization support[10, 9]. Currently we are working on various schemes to optimize and improve the performance of our one-sided implementation.

In this paper, we focus on two important aspects to improve the performance of active one sided communication. The first aspect is the ability to overlap multiple one sided operations which in turn can impact the latency. The second aspect is to better utilize the network link bandwidth by aggregating and re-ordering the communication operations within an window access epoch. Taking into consideration these aspects, we propose several schemes to schedule or reorder the RMA operations. In this process we also try to reduce the number of messages by aggregating the actual communication data with the synchronization message whenever possible. We have also come up with schemes

[*]This research is supported in part by Department of Energy's Grant #DE-FC02-01ER25506, National Science Foundation's grants #CCR-0204429 and #CCR-0311542, and grants from Mellanox and Intel.

that can take advantage of both point to point based one-sided implementation and direct one-sided implementation.

The effectiveness of these techniques have been evaluated across two different platforms equipped with PCI-X and PCI-Express bus, respectively. We have been able to improve the MPI_Put and MPI_Get latency by up to 44% and 42% respectively on PCI-Express platform. Further, we use certain common scenarios as case studies to demonstrate the potential of our schemes in real applications. And we are able to observe significant improvement in throughput.

The rest of the paper is organized in the following way. In Section 2, we provide an overview of the InfiniBand Architecture and describe the background for our work. In Section 3, we explain the motivation for our schemes. In section 4, we discuss detailed design issues. We evaluate our designs in section 5 and discuss the related work in section 6. Conclusions and future work are presented in section 7.

2. Background

2.1. MPI-2 one sided communication

In many parallel scientific applications the data distribution might be changing dynamically and the data access patterns could be irregular. For these kinds of applications, each process can compute what data it needs to access or update at other processes. But a process might not know which data in its local memory needs to be read or updated by other processes, and in some cases may not even know the identification of the remote processes. So in such situations, there might be only one side in the communication process which knows all the parameters needed to transfer the data. MPI-2 one sided communication specifically targets these kinds of communication patterns.

In MPI-2 one sided communication, the sender can access the remote address space directly. This one sided communication is also referred to as Remote Memory Access or RMA communication. In this model, the origin process (the process that issues the RMA operation) provides all the parameters needed for the communication. The memory area on the target that can be accessed by the origin process is called a Window. MPI-2 defines several different types of communication operations. They are MPI_Put, MPI_Get and MPI_Accumulate. MPI_Put and MPI_Get functions transfer the data to and from a window in a target process, respectively. The MPI_Accumulate function combines the data movement to target with a reduce operation.

By the semantics of one sided communication, when an one-sided call returns, the completion of the operation is not guaranteed. In order to make sure that the one-sided operation is finished, explicit synchronization operations must be used. In MPI-2, synchronization operations are classified as *active* and *passive*. Active synchronization involves both sides of communication while passive synchronization only involves the origin side. We mainly focus on active mode of synchronization in this paper.

The period between two synchronization calls is called as access epoch and exposure epoch on the origin and target process, respectively. MPI-2 semantics allows multiple communication calls during an access epoch. This is done to amortize the overhead of synchronization over multiple communication operations.

The MPI-2 semantics lays down certain rules which need to be followed for the MPI program to be semantically correct[12]. Some of these restrictions can be taken advantage of by a smart MPI-2 implementation to enhance the performance.

- A location in a window must not be accessed locally once an update to that location has started, until the update becomes visible in the private window copy in process memory.
- A location in a window must not be accessed as a target of an RMA operation once an update to that location has started, until the update becomes visible in the public window copy.

The above statements imply that within an access epoch there is no order between the different put and get communication calls. At the end of the synchronization all the communication operations must have completed, but the order in which they complete and when the actual communication occurs is undefined.

An MPI-2 implementation can thus exploit this flexibility to improve the performance. In this paper we come with several schemes to reorder these communication to improve the latency and throughput. We describe these schemes in detail in section 4.

2.2. InfiniBand Overview

The InfiniBand Architecture (IBA) [7] is an industry standard. It defines a switched network fabric for interconnecting processing nodes and I/O nodes. InfiniBand Architecture supports both channel semantics and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, InfiniBand provides Remote Direct Memory Access (RDMA) operations, including RDMA write and RDMA read. RDMA operations are one-sided and do not incur software overhead at the remote side. This fits in well with the MPI-2 semantics of one sided communication calls. In the current hardware, RDMA write has better performance than RDMA read. Compared with the send/receive operations, RDMA operations have several advantages. First the RDMA operations themselves are generally faster than send/receive operations because they are simpler at the hardware level. Second they do not involve managing and posting descriptors at the receiver side which would incur additional overheads and reduce the communication performance. VAPI is a Mellanox implementation of InfiniBand Verbs interface.

2.3. MVAPICH2

MVAPICH2 is our high performance implementation of MPI-2 over InfiniBand. The implementation is based on MPICH2. As a successor of MPICH[5], MPICH2[1] supports MPI-1 as well as MPI-2 extensions including one sided communication. We have been working on MVAPICH2 for the past several months and currently we are working on further optimizing the existing implementation for both two sided and one sided communication. There can be several different approaches for implementing one sided communication. One approach is based on top of point to point implementation provided by MPICH2. This approach involves the remote host involvement for communication and synchronization operations. In the second approach, the one sided operations were implemented at the CH3 level by extending the CH3 interface [10, 9]. This approach shows overall benefits in most cases with respect to latency and bandwidth. It also gives better overlap between computation and communication and also better scalability. Henceforth, we will refer to the first approach as *Point to Point Based* and second approach as *Direct One Sided*. Fig. 1 shows the path taken by both the approaches.

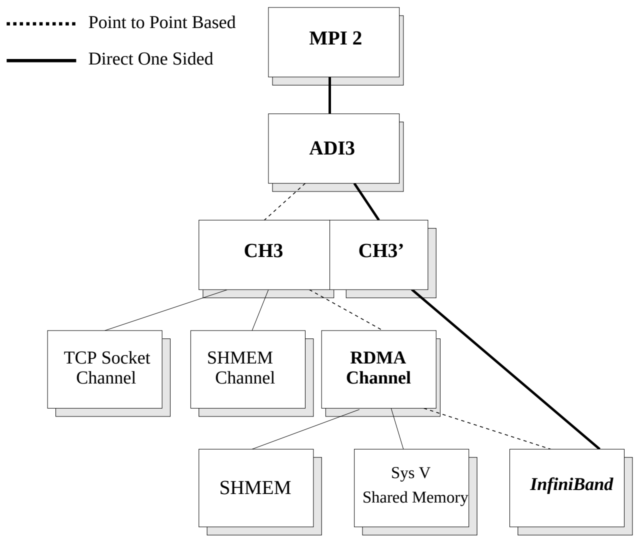


Figure 1. Implementations of one sided communication in MVAPICH2

3. Motivation

The motivation for our work here is two fold:

- As described in section 2.1, MPI-2 semantics does not impose any restrictions on when and in what order the RMA operations should occur within an access epoch. However both the current implementations (Point to Point Based and Direct One Sided) for active synchronization always maintain the order of the RMA operations. This might not always lead to the best or optimum usage of the underlying network capability. In this paper we want to exploit this flexibility to explore different ways to reorder these RMA operations based on the communication pattern to improve the latency, bandwidth and throughput.

- Message aggregation can reduce the latency for small RMA opearations because it can potentially reduce the number of messages. The Point to Point Based implementation can give this ability because of its two sided nature. With the Point to Point Based implementation several RMA operations can be reordered and combined/aggregated into a single message and the remote side can receive this combined message and scatter them. Aggregation of a RMA communication operation and a synchronization message is also feasible. Thus the Point to Point Based implementation can be leveraged to improve the performance of small messages.

4. Design and Implementation

As described in section 3, MPI-2 semantics potentially allow the implementation to reorder the actual completion of the RMA operations, such as MPI_Put and MPI_Get, issued during a window access epoch. Our main motivation is to utilize this flexibility to schedule these operations so that we can achieve better communication overlap, reduced latency and improved throughput on our InfiniBand implementation.

We propose two possible approaches for scheduling the RMA operations. The reordering approach focuses on reorganizing the MPI_Put and MPI_Get operations issued during a window access epoch to allow more efficient usage of network bandwidth. The aggregation approach tries to combine RMA operations to give better throughput.

4.1. Reordering approach

Since MPI-2 standard allows the actual communication for RMA operations to happen at synchronization time, we can hold all the RMA operations issued during a window access epoch until synchronization time. At this stage, we will have enough information of the communication pattern during this access period. Based on this information, we may re-order the issuing of these RMA operations to utilize the underlying InfiniBand network more efficiently.

4.1.1. Interleaving. The bidirectional bandwidth is always higher than the unidirectional bandwidth. This is because of the full usage of the link bandwidth of both directions. For example, with MVAPICH2 point to point communication, we are able to achieve 874MB/s peak unidirectional bandwidth while we can achieve 934MB/s in bidirectional bandwidth test. (The unit of bandwidth MB/s in this paper refers to Million bytes/sec) This trend is more obvious on PCI-Express systems because the bus contention is no longer the bottleneck in this scenario. The peak bandwidth number for unidirectional and bidirectional tests are 964MB/s and 1905MB/s on the PCI-Express system.

However, in a typical one sided communication scenario, only one direction of the link bandwidth is fully used, since the target side is not explicitly involved in the communication. But this does not mean that we can only stick with the highest possible unidirectional bandwidth provided by the link. For MPI_Put operations, we issue RDMA write operations at VAPI level to push the data out. The actual data flow is from the origin process to the target. But for MPI_Get operation, we issue RDMA read operation at VAPI level to fetch data from the remote side. So the actual data flow, especially for large size operations, is from the target process to the origin process.

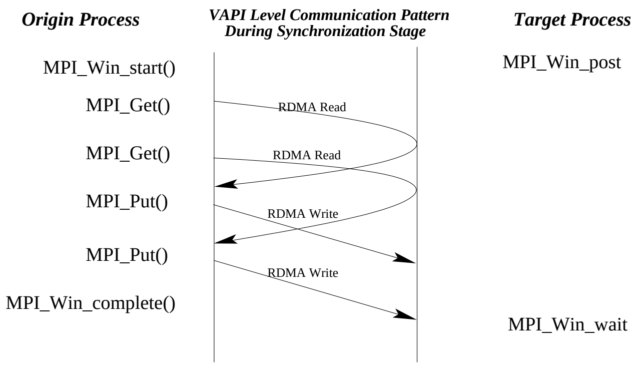


Figure 2. Sequential issue of MPI_Get and MPI_Put

Let us consider the following one sided communication patterns. In Fig. 2, the origin process issues several MPI_Get operations and then several MPI_Put operations during a RMA access epoch. In Fig. 3, the origin process issues the same number of MPI_Get and MPI_Put operations, but in an interleaved way. As we can observe, the second communication pattern in Fig. 3 can use the link bandwidth

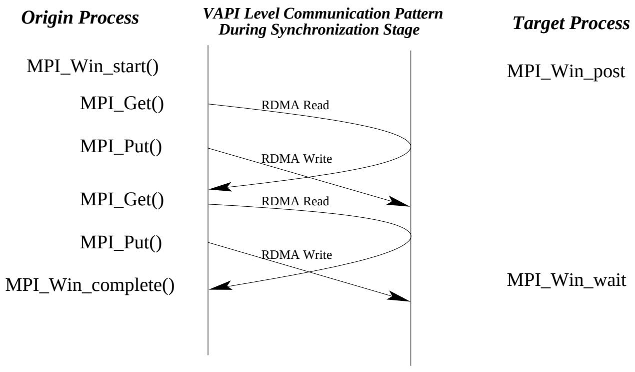


Figure 3. Interleaved issue of MPI_Get and MPI_Put

in a much more efficient way than the first communication pattern.

Though we know that the link bandwidth will be used more efficiently if the issuing of MPI_Put and MPI_Get is interleaved, we can not require the MPI programmer to understand this and always write the optimized program. But since the RMA operation can actually start during synchronization time, we can schedule the operations so that the corresponding VAPI level RDMA read and RDMA write operations are issued in a interleaved manner.

4.1.2. Prioritizing. One of the conclusions of our previous research is that the Direct One Sided implementation offers better latency than Point to Point Based implementation for large RMA operations. But it is still possible to further optimize the Direct One Sided implementation.

During the synchronization stage of direct one sided implementation, the origin process will issue a RDMA write to set a flag at the target process to indicate the end of the access epoch. Before that, if a MPI_Get operation was issued prior to the synchronization call, we need to wait for local completion of Get to ensure that the data has actually been fetched and ready for use by the end of synchronization phase.

During the access epoch, if the origin process calls several MPI_Put and MPI_Get operations, we want to give priority to MPI_Get operations in order to reduce the time involved in waiting for the local completion. Therefore we give priority to MPI_Get operations over MPI_Put operations. We first issue RDMA read required by MPI_Get and then issue RDMA write required by MPI_Put. Fig. 4 illustrates the potential benefits of our prioritizing scheme. It is to be noted that this prioritizing scheme does not necessarily contradict with the interleaving scheme we proposed in the last section. We can still interleave the operations but we can issue RDMA read operations first.

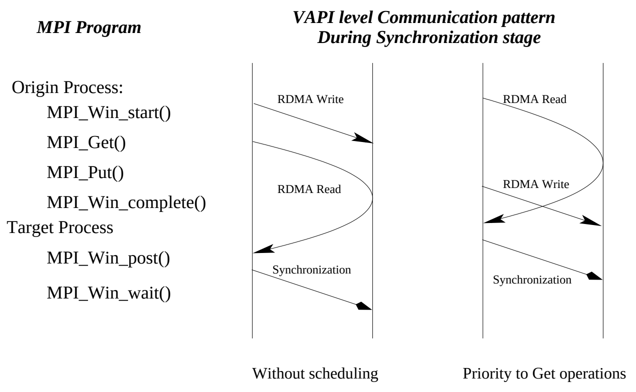


Figure 4. Potential benefit by giving priority to MPI_Get

4.2. Aggregation

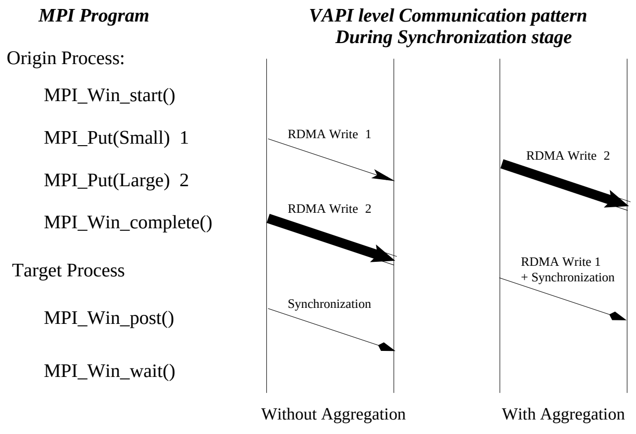


Figure 5. Aggregation of RMA operation and synchronization

As described earlier our goal here is to better utilize the network bandwidth. If we have multiple small RMA messages within an access epoch the network utilization would be suboptimal. Because, for small messages, the overhead associated with initiation and completion of RMA operations is relatively high.Hence a natural and obvious choice would be to try and see if we can aggregate several of these messages together. The users can use MPI user defined datatypes to aggregate several one sided and two sided operations to improve network utilization. However, our aim is to provide optimizations inside the MPI library so that we can deliver good performance even if there is no optimization at the user level. Also, as described in section 2, no order needs to be guaranteed among the MPI_Put/MPI_Get operations between two synchronization calls. So we are not violating any MPI-2 semantics by aggregating some of these operations, as long as all the data finally reaches the target side. We can consider the following two aggregation schemes:

- Aggregation between an RMA operation and a synchronization operation
- Aggregation between multiple RMA operations

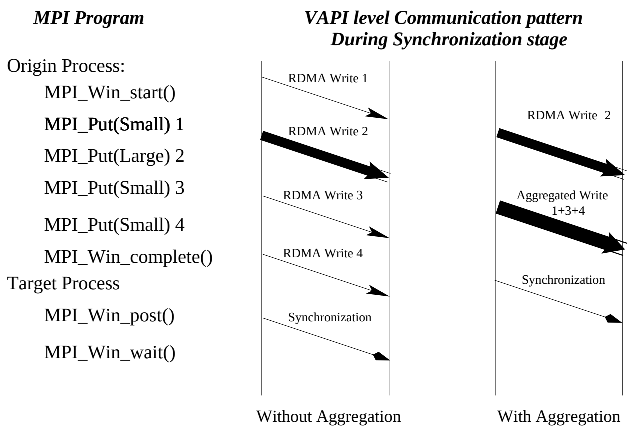


Figure 6. Aggregation of multiple small size RMA operations

These schemes are illustrated in Figs. 5 and 6. By utilizing Point to Point Based approach, we can aggregate multiple RMA operations or an RMA operation and a synchronization operation. In contrast, Direct One Sided approach cannot provide aggregation because the target is not involved in communication and hence cannot scatter aggregated messages into target buffers. To maximize aggregation, we defer small RMA messages until we have sufficiently large number of them. Then we can trigger deferred RMA messages as an aggregated operation and send it by Point to Point Based approach. Meanwhile, large size RMA operations are still issued by Direct One Sided approach. We can also hold back one small RMA operation and combine it with the synchronization operation. In this paper, we mainly focus on the aggregation between a RMA operation and a synchronization operation.

5. Performance Evaluation

In this section, we use several micro benchmarks to evaluate the performance of our different schemes.

Due to the lack of publicly available applications using MPI-2 one sided calls, we came up with our own benchmarks to evaluate our scheduling schemes. We use some specific throughput and latency tests to measure the impact of our re-ordering scheme. In addition to this, we use ping-pong latency tests for MPI_Put and MPI_Get to show the benefit of the aggregation scheme. These experiments have been conducted on two platforms specified in section 5.1.

5.1. Experimental Testbed

We evaluated our schemes on two different testbeds. The first testbed is equipped with PCI-X interface and the second is equipped with PCI-Express interface.

Our PCI-X testbed cluster consists of 8 SuperMicro SUPER X5DL8-GG nodes with ServerWorks GC LE chipsets, Intel Xeon 3.0 GHz processors based on IA32 architecture, and PCI-X 64-bit 133 MHz bus. The PCI-Express node of our testbed has a 3.4 GHz Intel Xeon processor based on EM64T architecture and runs in 64 bit mode with 8x PCI-Express interfaces. They are equipped with MT25208 HCAs with PCI-Express interfaces. On both platforms InfiniScale MTS2400 switch is used to connect all the nodes. The versions of InfiniBand SDK and firmware are 3.2 RC17 and 4.5.2 RC4-BUILD-001 respectively. The operating system used is RedHat Linux.

5.2. Impact of Re-ordering Scheme on different Communication Patterns

We created two communication patterns at microbenchmark level to study the impact of the re-ordering scheme we proposed in the previous section.

Communication Pattern 1: We created a throughput test which involves two processes. The first process starts a window access epoch and then issues 16 MPI_Put and 16 MPI_Get operations of the same size. The second process just starts an exposure epoch. The same sequence of operations are repeated for several iterations and we measure the maximum throughput we can achieve (in terms of MillionBytes/sec).

We compared the performance of re-ordering scheme and the original Direct One Sided implementation. On PCI-Express systems, as we can see from Fig. 7(a), with re-ordering scheme we are able to attain maximum throughput of 1788MB/s, which is much closer to the peak bidirectional bandwidth. We observe an improvement in throughput up to 76% compared with the original design. This trend is also there on IA32 systems where the maximum improvement of throughput is about 8%, as shown in Fig. 7(b). However, we do not get as much improvement as on EM64T testbed because on IA32 system, the PCI-X bus becomes the bottleneck.

Communication Pattern 2: The test consists of multiple iterations involving two processes. In each iteration, the first process calls MPI_Win_start to start a window access epoch, issues one MPI_Put and one MPI_Get, and then calls MPI_Win_complete to end the epoch. After that it starts and ends a window exposure epoch by calling MPI_Win_post and MPI_Win_wait. The second process does the same job, but in a reversed order, first it starts the exposure epoch then the access epoch. We measure the average latency for each iteration.

Our Scheduling scheme switches the order of these two operations when it is actually issuing the corresponding RDMA read or RDMA write during the access epoch. We can see that especially for large messages, we can show significant benefits by scheduling the operations internally. We can reduce the latency up to 40% on EM64T testbed and 20% on IA32 testbed, as shown in Fig. 8.

5.3. Impact of Aggregation Scheme on Latency

In this section we measure the impact of our aggregation scheme on MPI_Put and MPI_Get latency. The test consists of multiple iterations involving two processes. In each iteration, the first process starts a window access, issues a RMA operation (MPI_Put or MPI_Get) and then ends the epoch. Then it starts and ends a window exposure epoch. The second process does the same job, but in a reversed order. We measure the time needed for each iteration and define half of its value as the ping-pong latency for the RMA operation.

Fig. 9(a) compares the ping-pong latency for MPI_Put operation and Fig. 10(a) compares the ping-pong latency for MPI_Get operations on EM64T testbed. The aggregation scheme did noticeably better than our original Direct One Sided implementation for small size RMA operation. We see an improvement of up to 44% for MPI_Put latency and 42% for MPI_Get latency. For larger sizes, the aggregation scheme actually falls back to Direct One Sided implementation so that these two schemes delivers the same latency. We can observe the similar trends on IA32 platform, as shown in Fig. 9(b) and Fig. 10(b). The maximum improvement is around 38% and 42% for MPI_Put and MPI_Get latency respectively.

6. Related Work

There are several studies regarding implementing one sided communication in MPI-2. Some of the MPI-2 implementations which implement one sided communication are WMPI [13], NEC [8] and SUN-MPI [3]. We are aware of MPICH2 performing aggregation between the last one sided operation with a synchronization. In [17], reordering of one sided operations is done to reduce the cost of lock synchronization operation.

Besides MPI, there are other programming models that use one sided communication. ARMCI [15], GASNET [2] and BSP [4] are some examples of this model. One distinguishing feature of MPI as compared to these is that MPI supports both one sided and two sided communications, which we use to our advantage in implementing our

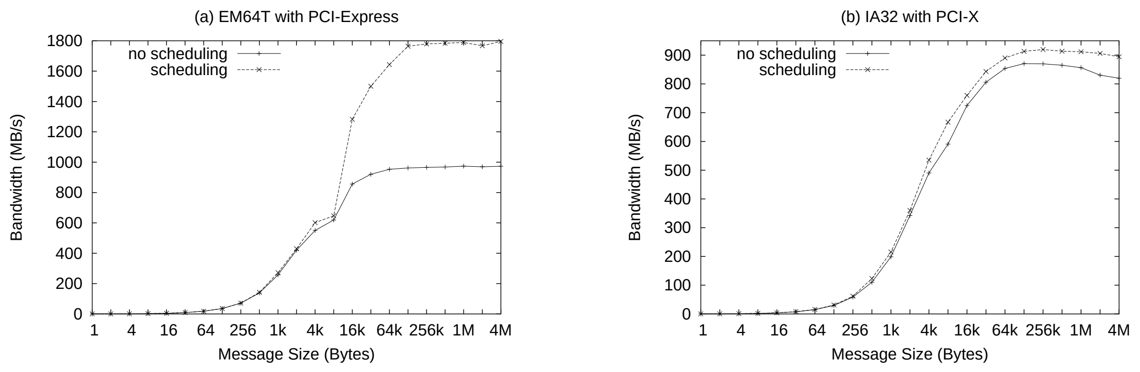


Figure 7. Impact of scheduling on throughput on EM64T and IA32

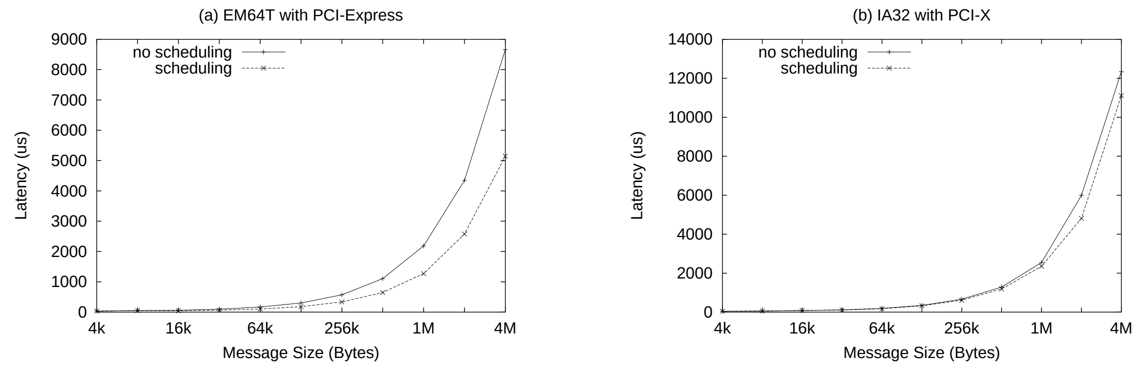


Figure 8. Impact of scheduling on latency on EM64T and IA32

schemes. It is to be noted that ARMCI performs aggregation [16].

7. Conclusions and Future work

This paper describes different scheduling and aggregation schemes to improve the performance of MPI-2 one sided communication over InfiniBand. We have proposed different schemes and evaluated them with micro benchmarks and different scenarios on two different platforms. With scheduling scheme, we see an improvement in the throughput up to 76%, latency up to 40% for certain scenarios. With aggregation scheme, we observe an improvement of 44% and 42% for MPI_Put and MPI_Get latency on PCI-Express platform. Similar trends were observed for PCI-X platform.

As part of our future work, we will extend our implementation of our aggregation scheme for combining multiple RMA operations. We would like to explore more optimized scheduling schemes. Further, we intend to merge the different schemes into one framework which can adaptively choose based on the communication pattern.

References

[1] Argonne National Laboratory. MPICH2. http://www-unix.mcs.anl.gov/mpi/mpich2/.

[2] D. Bonachea. GASNet Specifi cation, v1.1. Technical Report UCB/CSD-02-1207, Computer Science Division, University of California at Berkeley, October 2002.

[3] S. Booth and F. E. Mourao. Single Sided MPI Implementations for SUN MPI. In *Supercomputing*, 2000.

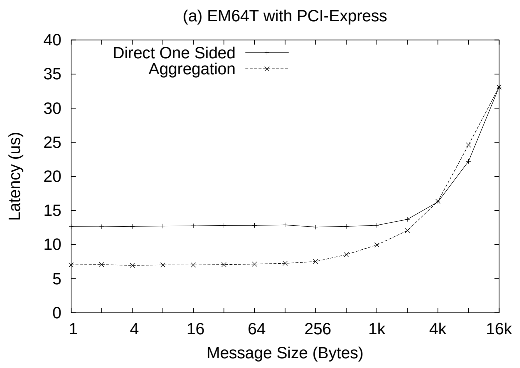
[4] M. Goudreau, K. Lang, S. B. Rao, T. Suel, and T. Tsantilas. Portable and Effcient Parallel Computing Using the BSP Model. *IEEE Transactions on Computers*, pages 670–689, 1999.

[5] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[6] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd edition. MIT Press, Cambridge, MA, 1999.

[7] Infi niBand Trade Association. Infi niBand Architecture Specifi cation, Release 1.0, October 24 2000.

[8] J. Traff and H. Ritzdorf and R. Hempel. The Implementation of MPI-2 One-Sided Communication for the NEC SX. In *Proceedings of Supercomputing*, 2000.



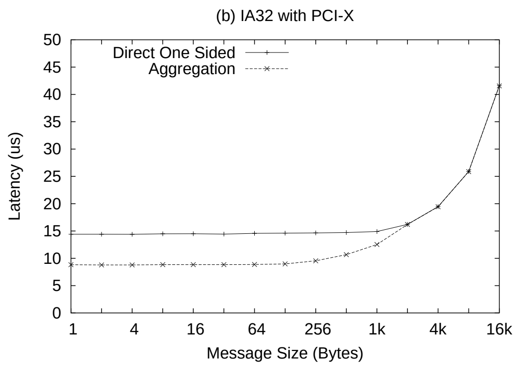
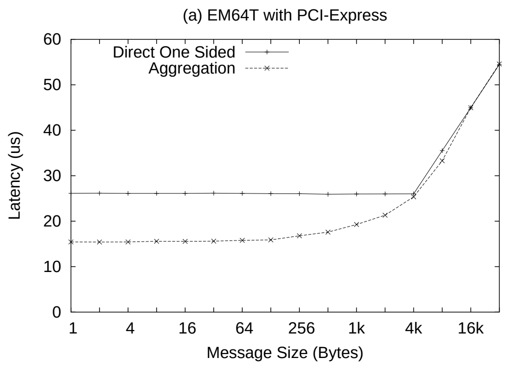


Figure 9. One sided MPI_Put latency on EM64T and IA32



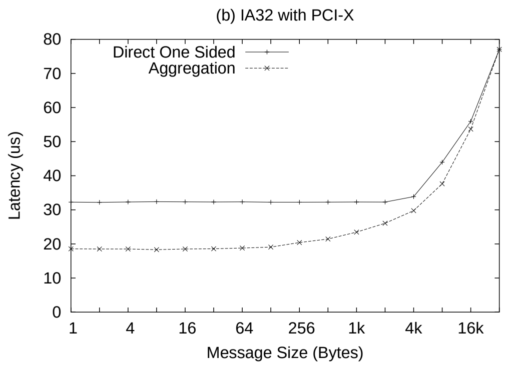


Figure 10. One sided MPI_Get latency on EM64T and IA32

[9] W. Jiang, J.Liu, H. W. Jin, D. K. Panda, D. Buntinas, R.Thakur, and W.Gropp. Effi cient Implementation of MPI-2 Passive One-Sided Communication on Infi niBand Clusters. EuroPVM/MPI, September 2004.

[10] J. Liu, W. Jiang, H.-W. Jin, D. K. Panda, , W. Gropp, and R. Thakur. High Performance MPI-2 One-Sided Communication over Infi niBand. International Symposium on Cluster Computing and the Grid (CCGrid 04), April 2004.

[11] J. Liu, W. Jiang, P. Wyckoff, D. K. Panda, D. Ashton, D. Buntinas, W. Gropp, and B. Toonen. Design and Implementation of MPICH2 over Infi niBand with RDMA Support. Int'l Parallel and Distributed Processing Symposium (IPDPS 04), April 2004.

[12] Message Passing Interface Forum. MPI-2: A Message Passing Interface Standard. *High Performance Computing Applications*, 12(1–2):1–299, 1998.

[13] F. E. Mourao and J. G. Silva. Implementing MPI's One-Sided Communications for WMPI. In *EuroPVM/MPI*, September 1999.

[14] Network-Based Computing Laboratory. MPI over Infi niBand Project. http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html.

[15] J. Nieplocha and B. Carpenter. ARMCI: A Portable Remote Memory Copy Library for Distributed Array Libraries and Compiler Run-Time Systems. *Lecture Notes in Computer Science*, 1586, 1999.

[16] J. Nieplocha, V. Tipparaju, M. Krishnan, G. Santhanaraman, and D. K. Panda. Optimizing Mechanisms for Latency Tolerance in Remote Memory Access Communication on Clusters . In *Proceedings of the IEEE International Conference on Cluster Computing*, 2003.

[17] R. Thakur, W. Gropp, and B. Toonen. Minimizing Synchronization Overhead in the Implementation of MPI One-Sided Communication. In *EuroPVM/MPI*, September 2004.