High Performance MPI-2 One-Sided Communication over InfiniBand*

Weihang Jiang Jiuxing Liu Hyun-Wook Jin Dhabaleswar K. Panda
William Gropp† Rajeev Thakur†

Computer and Information Science
The Ohio State University
Columbus, OH 43210
{jiangw, liuj, jinhy, panda}@cis.ohio-state.edu

†Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{gropp, thakur}@mcs.anl.gov

Abstract

Many existing MPI-2 one-sided communication implementations are built on top of MPI send/receive operations. Although this approach can achieve good portability, it suffers from high communication overhead and dependency on remote process for communication progress. To address these problems, we propose a high performance MPI-2 one-sided communication design over the InfiniBand Architecture. In our design, MPI-2 one-sided communication operations such as MPI_Put, MPI_Get and MPI_Accumulate are directly mapped to InfiniBand Remote Direct Memory Access (RDMA) operations.

Our design has been implemented based on MPICH2 over InfiniBand. We present detailed design issues for this approach and perform a set of micro-benchmarks to characterize different aspects of its performance. Our performance evaluation shows that compared with the design based on MPI send/receive, our design can improve throughput up to 77%, and reduce lantency and synchronization overhead up to 19% and 13%, respectively. Under certain process skew, the bad impact can be significantly reduced by new design, from 41% to nearly 0%. It also can achieve better overlap of communication and computation.

1 Introduction

In the area of high performance computing, MPI [9] has been the *de facto* standard for writing parallel applications. The original MPI standard (MPI-1) specifies a message passing communication model based on send and receive operations. In this model, communication involves both sender and receiver sides, and the synchronization is achieved implicitly through communication operations. This model is also called *two-sided communication*.

As an extension to MPI-1, the MPI-2 [14] standard introduces the *one-side communication* model. In this model, one process specifies all communication parameters, and the synchronization is done explicitly to ensure the completion of communication. One-sided communication operations in MPI-2 include MPI_Put, MPI_Get and MPI_Accumulate.

One common way to implement MPI-2 one-sided communication is to use existing MPI two-sided communication operations such as MPI_Send and MPI_Recv. This approach has been used in several popular MPI implementations, including the current MPICH2 [1] and SUN MPI [5]. Although this approach can improve portability, it has some potential problems:

- Protocol overhead: Two-sided communication operations incur many overheads such as memory copy, matching of send and receive operations and handshake in Rendezvous protocol. These overheads decrease communication performance for one-sided communication.

- Dependency on remote process: The communication progress of one-side communication operations are dependent on the remote process in this approach. As a result, process skew may significantly degrade communication performance.

Recently, InfiniBand [11] has entered the high performance computing market. InfiniBand architecture supports Remote Direct Memory Access (RDMA). RDMA operations enable direct access to the address space of a remote process and their semantics match quite well with MPI-2

*This research was supported in part by Department of Energy's Grant #DE-FC02-01ER25506, and National Science Foundation's grants #CCR-0204429 and #CCR-0311542. This work was also supported in part by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

one-sided communication. In our recent work [13], we have proposed a design which uses RDMA to implement MPI two-sided communication. However, since its MPI-2 one-sided communication operations are implemented on top of two-sided communication, this implementation still suffers from the problems mentioned above.

In this paper, we propose using InfiniBand RDMA operations directly to provide efficient and scalable one-sided communication in MPI-2. Our RDMA based design maps MPI_Put and MPI_Get directly to InfiniBand RDMA write and RDMA read operations. Therefore, one-sided communication can avoid the protocol overhead in MPI send and receive operations. Since RDMA write and RDMA read in InfiniBand are transparent to the remote side, MPI one-sided communication can make progress without its participation. Therefore, our design is less prone to process skew and also allows better communication/computation overlap.

In this work, we present detailed design issues in our RDMA based approach. We have implemented our design based on our MPICH2 implementation over InfiniBand. We use a set of micro-benchmarks to evaluate its performance. These micro-benchmarks characterize different aspects of MPI-2 one-sided communication, including communication performance, synchronization overhead, dependency on remote process, communication/computation overlap and scalability. Our performance evaluation shows that the RDMA based design can bring improvements in all these aspects. It can improve bandwidth up to 77% and reduce latency and synchronization overhead up to 19% and 13%, respectively. Under certain process skew, the bad impact can be significantly reduced by new design, from 41% to nearly 0%.

The remaining part of the paper is organized as follows: In Section 2, we provide background information about MPI-2 one-sided communication and InfiniBand. In Section 3, we introduce the current design for MPI-2 one-sided communication in MPICH2. We describe our design in Section 4. In Section 5, we present performance evaluation results. We discuss related work in Section 6 and conclude the paper in Section 7.

2 Background

2.1 MPI-2 One-Sided Communication

In MPI-2 one-sided communication model, a process can access another process's memory address space directly. Unlike MPI two-sided communication in which both sender and receiver are involved for data transfer, one-sided communication allows one process to specify all parameters for communication operations. As a result, it can avoid explicit coordination between the sender and the receiver. MPI-2 defines *origin* as the process that performs the one-sided communication, and *target* as the process in which the memory is accessed. A memory area on target to which origin can access through the one-sided communication is called a *window*. Several one-sided communication operations are defined in MPI-2. 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 process with a reduce operation.

It should be noted that returning from communication operations such as MPI_Put does not guarantee the completion of the operations. To make sure an one-sided operation is finished, explicit synchronization operations must be used. In MPI-2, synchronization operations are classified as *active* and *passive*. The active synchronization involves both sides of communication while passive synchronization only involves the origin side. In Figure 1, we show an example of MPI-2 one-sided communication with active synchronization. We can see that synchronization is achieved through four MPI functions: MPI_Win_start, MPI_Win_complete, MPI_Win_post and MPI_Win_wait. MPI_Win_post and MPI_Win_wait specify an *exposure epoch* in which other processes can access a memory window in this process. MPI_Win_start and MPI_Win_complete specify an *access epoch* in which the current process can use one-side communication operations such as MPI_Put to access memory in the target process. Multiple operations can be issued in the access epoch to amortize the overhead of synchronization. The completion of these operations are not guaranteed until the MPI_Win_complete returns. The active synchronization can also be achieved through MPI_Win_fence.

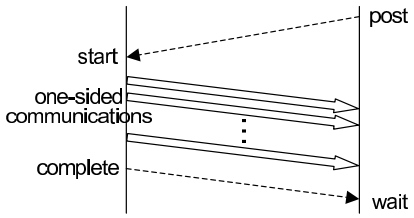

Figure 1. MPI-2 One-Sided Communication with Active Synchronization

2.2 InfiniBand

The InfiniBand Architecture is an industry standard which defines communication and infrastructure for inter-processor communication and I/O. InfiniBand supports both channel and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, InfiniBand supports RDMA operations such as RDMA write and RDMA read. RDMA operations are one-sided and do not incur software overhead at the other side. In these operations, memory buffers must first be

registered before they can be used for communication. Then the sender (initiator) starts RDMA by posting an RDMA descriptor which contains both the local data source addresses (multiple data segments can be specified at the source) and the remote data destination address. The operation is transparent to the software layer at the receiver side. We should note that the semantic of InfiniBand RDMA operations is similar to that of MPI-2 one-sided communication. Therefore, it is expected that if we implement the one-sided communication with RDMA operations, we can achieve high performance and offload the communication from the target completely.

InfiniBand also provides remote atomic operations such as Compare-and-Swap and Fetch-and-Add. These operations can read and then change the content of a remote memory location in an atomic manner.

3 Send/Receive Based MPI-2 One-Sided Communication Design

As we have mentioned, MPI-2 one-sided communication can be implemented using MPI two-sided communication operations such as MPI_Send, MPI_Recv and their variations (MPI_Isend, MPI_Irecv, MPI_Wait, etc.). (In the following discussions, we use "send" and "receive" to refer to different forms of MPI_Send and MPI_Recv, respectively.) We call this *send/receive based* approach. The current MPICH2 implementation uses such an approach. In this section, we will discuss MPICH2 as an implementation example of one-sided communication.

MPICH [8], developed by Argonne National Laboratory, is one of the most popular MPI implementations. The original MPICH provides supports for MPI-1 standard. As a successor of MPICH, MPICH2 [1] supports not only MPI-1 standard, but also MPI-2 extensions such as one-sided communication, dynamic process management, and MPI I/O. Although MPICH2 is still under development, beta versions are already available for developers. Our discussion is based on MPICH2 over InfiniBand (MVAPICH2)[1] [13].

3.1 Communication Operations

For the MPI_Put operation, the origin process first sends information about this operation to the target. This information includes target address, data type information, etc. Then the data itself is also sent to the target. After receiving the operation information, the target uses another receive operation for the data. In order to perform the MPI_Get operation, first the origin process sends a request to the target, which informs it the data location, data type and length. After receiving the request, the target process sends the requested data to the origin process. The origin finishes the operation by receiving the data to its local buffer.

[1]The current MVAPICH2 implementation is based on MPICH2 version 0.96p1. MVAPICH2 is available from [17].

For MPI_Accumulate, the origin process uses a similar approach to send the data to the target process. Then the target receives the data and performs a local reduce operation.

The send/receive based approach has very good portability. Since it only depends on MPI two-sided communication, its implementation is completely platform independent. However, it also has many drawbacks. First, it suffers from high protocol overhead in MPI send/receive operations. For example, MPI send/receive operations uses Rendezvous protocol for large messages. In order to achieve zero-copy, the current MPICH2 uses a handshake in the Rendezvous protocol to exchange buffer addresses. However, since in one-sided communication, target buffer address information is available at the origin process, this handshake is unnecessary and brings degradation of communication performance. In addition, MPI send/receive may involve other overheads such as send/receive matching and extra copies.

Since the target is actively involved in the send/receive based approach, the overhead at the target process increases. The target process may become a performance bottleneck because of this increased overhead.

The send/receive based approach also makes the origin process and the target process tightly coupled in communication. The communication of origin process depends heavily on the target to make progress. As a result, process skew between the target and the origin may significantly degrade communication performance.

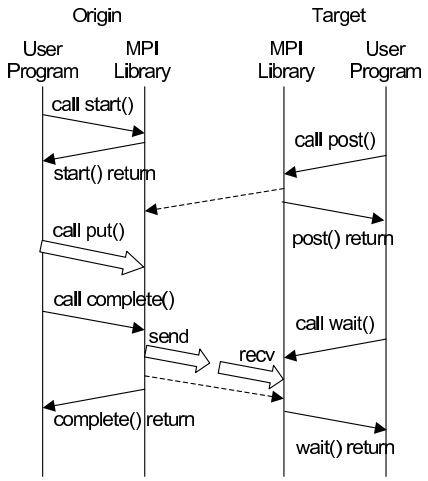

Figure 2. Send/Receive Based One-Sided Communication Implementation

3.2 Synchronization Operations

MPICH2 implements the active synchronization for the one-sided communication, the passive synchronization is still under development. Therefore, we focus on active synchronization in this paper.

In MPI-2 one-side communication, synchronization can be done using MPI_Win_start, MPI_Win_complete, MPI_Win_post and MPI_Win_wait. At the origin side, communication is guaranteed to finish only after MPI_Complete. Therefore, implementors have a lot of flexibility with respect to when to carry out the actual communication. In the send/receive based approach, communication involves both sides. Since the information about the communication is only available at the origin side, the target needs to be explicitly informed about this information. One way to address this problem in send/receive based approaches is to delay communication until MPI_Win_complete. Within this function, the origin sends information about all possible operations. In MPI_Win_wait, the target receives this information and takes appropriate actions. An example of send/receive based implementation is shown in Figure 2.

Delayed communication used in send/receive based approach allows for certain optimizations such as combining of small messages to reduce per-message overhead. However, since the actual communication does not start until MPI_Complete, the communication cannot be overlapped with computation done in the access epoch. This may lead to degraded overall application performance.

In the current MPICH2 design, the actual communication starts when there are enough one-sided communication operations posted to cover the cost of synchronization. This design can potentially take advantage of the optimization opportunities in delayed communication and also allow for communication/computation overlap. However, since one-sided communication is built on top of send/receive, the actual overlap depends on how the underlying send/receive operations are implemented. In many MPI implementations, sending/receiving a large message goes through Rendezvous protocol, which needs host intervention for a handshake process before the data transfer. In these cases, good communication/computation overlap is difficult to achieve.

4 RDMA Based MPI-2 One-Sided Communication Design

As we have described in Section 3, one-sided communication in MPICH2 is currently implemented based on MPI send/receive operations. Therefore, it still suffers from the limitation of the two-sided communication design even though the MVAPICH2 [13]. In this section, we discuss how to utilize InfiniBand features such as RDMA operations to address these potential problems. MPICH2 has a flexible layered architecture in which implementations can be done at different levels. Our MVAPICH2 implementation over InfiniBand [13] [17] was done using the RDMA Channel Interface. However, this interface currently does not provide us with direct access to all the RDMA and atomic functions in InfiniBand. To address this issue, we use a customized interface to expose these functionalities to the upper layer and implement our design directly over this interface. The basic structures of our design and the original design are shown in Figure 3.

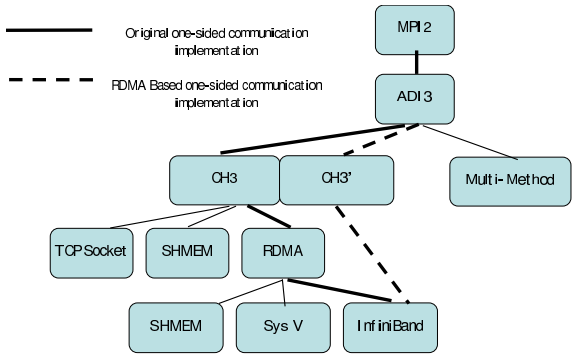

Figure 3. Design Architecture

4.1 Communication Operations

We implement the MPI_Put operation with RDMA write. Through exchanging memory addresses at window creation time, we can keep record of all target memory addresses on all origin processes. When MPI_Put is called, an RDMA write operation is used, which directly transfers data from memory in the origin process to remote memory in the target process, without involving the target process. The MPI_Get operation is implemented with the RDMA read operation in InfiniBand. Based on InfiniBand RDMA and atomic operations, we have designed the accumulate operation as follows: The origin fetches the remote data from target using RDMA read, performs a reduce operation, and updates remote data by using RDMA write. Since there may be more than one origins, we use the Compare-and-Swap atomic operation to ensure mutual exclusive access.

By using RDMA, we can avoid protocol overhead of two-sided communication. For example, the handshake in Rendezvous protocol is avoided. Also, the matching between send and receive operations is no longer needed. So the overhead associated with unexpected/expected message queue maintenance, tag matching and flow control is eliminated.

More importantly, the dependency on remote process for communication progress is reduced. Unlike the send/receive based approach, using RDMA operations directly does not involve the remote process. Therefore, the communication can make progress even when the remote process is doing computation. As a result, our implementation suffers much less from process skew. Moreover, our design exploiting RDMA operations can make implementation of passive one-sided communication much easier because the target is not required to respond to one-sided communication operations.

4.2 Synchronization Operations

In some send/receive based designs, actual communication is delayed until MPI_Win_complete is called. In our design, the one-sided communication will start as soon as the post operation is called. In our implementation, the origin process maintains a bit vector. Each bit represents the status of a target. A target uses RDMA write to change the corresponding bit. By checking the bits, the origin process can get synchronization information and start communications.

Targets can not leave MPI_Win_wait until communication has been finished. Therefore origin processes need to inform the targets after they have completed communication. For this purpose we also use RDMA write to achieve better performance. Before leaving the MPI_Win_wait operation, the targets check to make sure all origin processes have completed communication. An example of this RDMA based implementation is shown in Figure 4.

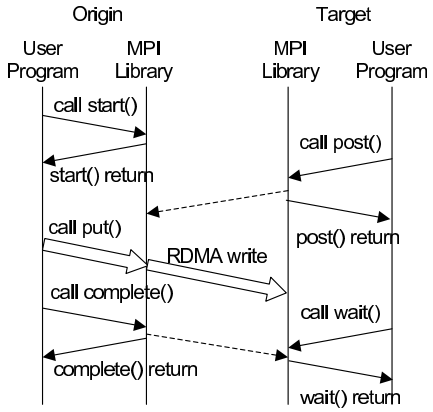

Figure 4. RDMA Based One-Sided Communication Implementation

4.3 Other Design Issues

By using RDMA, we potentially can achieve better performance. However, it also introduces several design challenges.

An important issue we should consider in exploiting RDMA operations is the memory registration. Before performing any RDMA operation, both source and destination data buffers need to be registered. The memory registration is an expensive operation. Therefore, it can degrade communication performance significantly if done in the critical path. All memory buffers for the one-sided communication on the target processes are declared when the window is created. Thus, we can avoid memory registration overheads by registering the memory buffers at the window creation time. For memory buffers at the origin side, pin-down cache [10] is used to avoid registration overhead for large messages. For small messages, we copy them to a pre-registered buffer to avoid registration cost.

Another important issue is to handle user-defined data type. The original approach requires data type processing at the target side. With RDMA operations, we can avoid this overhead by initiating multiple RDMA operations. Currently, our design only deals with simple data types. For complicated non-contiguous data types, we fall back on the original send/receive based implementation.

5 Performance Evaluation

In this section, we perform a set of micro-benchmarks to evaluate the performance of our RDMA based MPI-2 one-sided communication design and compare them with the original design in MPICH2. We have considered various aspects of MPI-2 one-sided communication such as synchronization overhead, data transfer performance, communication and computation overlap, dependency on remote process and scalability with multiple origin processes.

We focus on active one-sided communication in the performance evaluation. Our tests use MPI_Win_start, MPI_Win_complete, MPI_Win_post and MPI_Win_wait functions for synchronization. However, most of the conclusions in this section are also applicable to programs using MPI_Win_fence.

5.1 Experimental Testbed

Our experimental testbed consists of a cluster system with 8 SuperMicro SUPER P4DL6 nodes. Each node has dual Intel Xeon 2.40 GHz processors with a 512K L2 cache and a 400 MHz front side bus. The machines are connected by Mellanox InfiniHost MT23108 DualPort 4X HCA adapter through an InfiniScale MT43132 Eight 4x Port InfiniBand Switch. The HCA adapters work under the PCI-X 64-bit 133MHz interfaces. We used the Linux Red Hat 7.2 operating system with 2.4.7 kernel. The compilers we used were GNU GCC 2.96 and GNU FORTRAN 0.5.26.

5.2 Latency

For MPI two-sided communication, a ping-pong latency test is often used to characterize its performance. In this subsection, we use a similar test for MPI-2 one-side communication. The test consists of multiple iterations using two processes. Each iteration consists of two epochs. In the first epoch, the first process does an MPI_Put operation. In the second epoch, the second process does an MPI_Put operation. We then report the time taken for each epoch.

Figure 5 compares the ping-pong latency of our RDMA based design with the original MPICH2. We can see that the RDMA based approach can improve the latency. For small messages, it reduces latency from 15.6μs to 12.6μs (19% improvement). For large messages, since the handshake in

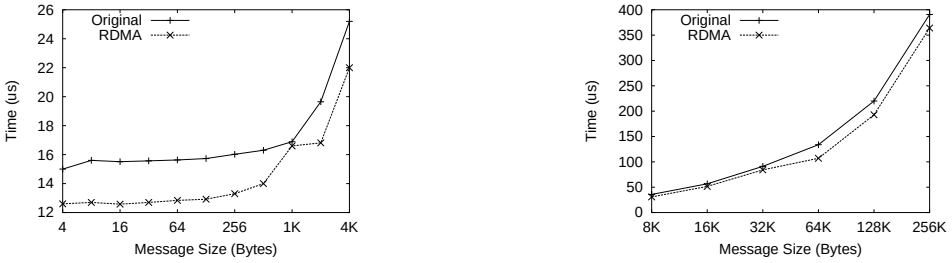

Figure 5. Ping-Pong Latency

Rendezvous protocol is avoided, it also gives better performance. The improvement is up to 17μs.

A bi-directional latency test is often used to compare the performance of one-sided communication to two-sided communication. In this test, both sides send a message to the other side. In the one-sided version, the test is done using MPI_Put and MPI_Win_fence. In the two-sided version, the test is done using MPI_Isend, MPI_Irecv and MPI_Waitall. Figure 6 shows the performance results. We can observe that for very small messages, two-sided communication performs better because it does not use explicit synchronization. For one-sided communication, our RDMA based design always performs better than the original design. For messages larger than 4KB, it even outperforms two-sided communication.

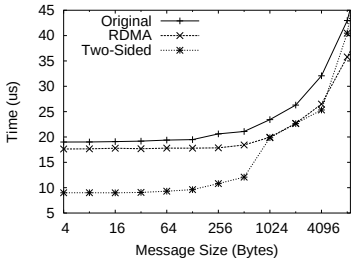

Figure 6. Bi-Directional Latency

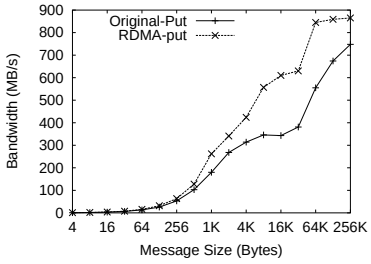

Figure 7. MPI_Put Bandwidth

5.3 Bandwidth

In applications using MPI-2 one sided operations, usually multiple communication operations are issued in each epoch to amortize the synchronization overhead. Our bandwidth test can be used to characterize performance in this

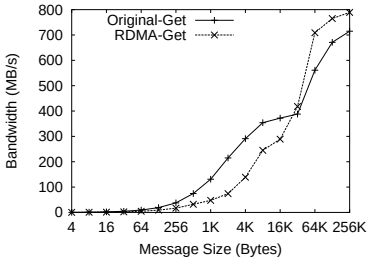

Figure 8. MPI_Get Bandwidth

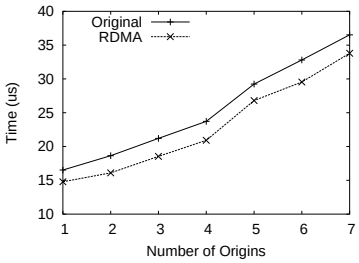

Figure 9. Synchronization Overhead

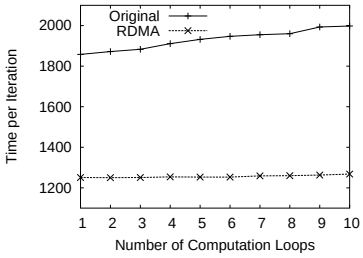

Figure 10. Overlap Test Results

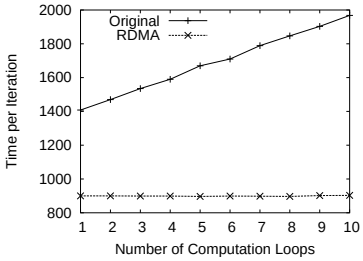

Figure 11. Process Skew Test Results

scenario. This test consists of multiple epochs. In each epoch, *W* MPI_Put or MPI_Get operations are issued where *W* is a pre-defined burst size.

Figures 7 and 8 show the bandwidth results of MPI_Put and MPI_Get with a burst size(*W*) of 16. We can see that the RDMA based approach always performs better for MPI_Put. The improvement can be up to 77% for certain message size. For 256KB messages, it delivers a bandwidth of 865MB/. (Note that unless stated otherwise, the unit MB in this paper is an abbreviation for 10^6 bytes.)

However, we also observe that the RDMA based approach does not perform as well as the original approach for MPI_Get with small messages. This is because RDMA read is used in our new design for MPI_Get while the original approach uses RDMA write. The bandwidth drop is due to the performance difference between InfiniBand RDMA read and RDMA write.

5.4 Synchronization Overhead

In MPI-2 one-sided communication, synchronization must be done explicitly to make sure data transfer has been finished. Therefore, the overhead of synchronization has great impact on communication performance. To characterize this overhead, we use a simple test which calls only MPI-2 synchronization functions (MPI_Win_start, MPI_Win_complete, MPI_Win_post and MPI_Win_wait) for multiple iterations. The test is done using one target process with multiple origin processes.

Figure 9 shows the time taken for each iteration for the original design and our RDMA based design. We can see that our new design slightly reduces synchronization time. When there is one origin, synchronization time is reduced from 16.52 microseconds to 14.78 microseconds (13% improvement). This is because we use InfiniBand level RDMA operations instead of calling MPI send and receive functions for synchronization. We have also done the test for one origin process with multiple target processes and the results are similar to Figure 9.

5.5 Communication/Computation Overlap

As we have mentioned, by using RDMA, we can possibly achieve better overlapping of communication and computation, which may lead to improved application performance. In this subsection, we have designed an overlap test to measure the ability to overlap communication and computation for different one-sided communication implementations.

The overlap test is very similar to the bandwidth test. The difference is that we have inserted a number of computation loops after each communication operation. Each computation loop increases a counter for 1,000 times. Figure 10 shows how the average time for one iteration of the test changes when we increase the number of computation loops for 64KB messages. We can see that the RDMA based design allows overlap of communication and computation and therefore its performance is not affected by increasing computation time. However, the original design shows lower performance when the computation increases.

5.6 Impact of Process Skew

As we have discussed, one of the advantages of using InfiniBand RDMA to implement MPI-2 one-sided communication is that the communication can make progress without depending on the target process. Therefore, skew between the origin and the target process will have less impact on the communication performance. Our process skew test is based on the bandwidth test. Process skew is emulated by adding different number of computation loops (with each loop increasing a counter for 10,000 times) between MPI_Win_post and MPI_Win_wait in the target process.

Figure 11 shows the performance results for 64KB messages. We can see that process skew does not affect the RDMA based approach at all. However, the performance of the original design drops significantly with the increase of process skew.

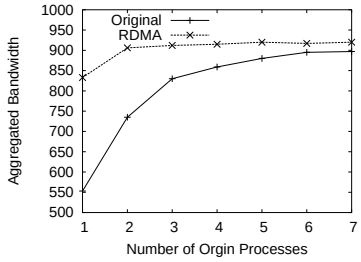

Figure 12. Aggregated Bandwidth with Multiple Origin Processes

5.7 Performance with Multiple Origin Processes

Scalability is very important for MPI-2 designs. In MPI-2 one-sided communication, it is possible for multiple origin processes to communicate with a single target process. Figure 12 shows the aggregated bandwidth of all origin processes in this scenario. Here we use 64KB as message size and 16 as burst size(*W*). We should note that the aggregated bandwidth is limited by the PCI-X bus at the target node. We can observe that since the RDMA based design incurs very little overhead at the target process, it reaches a peak bandwidth of over 920MB/s even with a small number of origin processes. The original design can only deliver a maximum bandwidth of 895MB/s.

6 Related Work

Besides MPI, there are other programming models that uses one-sided communication. Some of the examples are

ARMCI [16], BSP [7] and GASNET [4]. These programming models use one-sided communication as the primary communication approach while in MPI, both one-sided and two-sided communication are supported.

There have been studies regarding implementing one-sided communication in MPI-2. Similar to the current MPICH2, work in [5] describes an implementation based on MPI two-sided communication. MPI-2 one-sided communication has also been implemented by taking advantage of globally shared memory in some architectures [12, 15]. For distributed memory systems, some of the existing studies have exploited the ability of remotely accessing another process's address space provided by the interconnect to implement MPI-2 one-sided operations [2, 18, 3]. In this paper, our target architecture is InfiniBand, which provides very flexible RDMA as well as atomic operations. We focus on the performance improvement of using these operations compared with the send/receive based approach.

Work in [6] provides a performance comparison of several existing MPI-2 implementations. They have used a ping-pong benchmark to evaluate one-sided communication. However, their results do not include the MPICH2 implementation. In this paper, we focus on MPICH2 and introduce a suite of micro-benchmarks which provide a more comprehensive analysis of MPI-2 one-sided operations, including communication and synchronization performance, communication/computation overlap, dependency on remote process and scalability.

7 Conclusions and Future Work

In this paper, we have proposed a design of MPI-2 one-sided communication over InfiniBand. This design eliminates the involvement of targets in one-sided communication completely by utilizing InfiniBand RDMA operations.

Through performance evaluation, we have shown that our design can achieve lower overhead and higher communication performance. Moreover, experimental results have shown that the RDMA based approach allows for better overlap between computation and communication. It also achieves better scalability with multiple number of origin processes.

As future work, we are working on applying the RDMA approach also to the passive synchronization. We expect that the RDMA approach can give similar benefits in implementing the passive synchronization. Another direction we are currently pursuing is better support for non-contiguous data type in one-sided communication.

References

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

[2] N. Asai, T. Kentemich, and P. Lagier. MPI-2 Implementation on Fujitsu Generic Message Passing Kernel. In *SC*, 1999.

[3] M. Bertozzi, M. Panella, and M. Reggiani. Design of a VIA Based Communication Protocol for LAM/MPI Suite. In *9th Euromicro Workshop on Parallel and Distributed Processing*, September 2001.

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

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

[6] E. Gabriel, G. E. Fagg, and J. J. Dongarra. Evaluating the Performance of MPI-2 Dynamic Communicators and One-Sided Communication. In *EuroPVM/MPI*, September 2003.

[7] 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.

[8] 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.

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

[10] H. Tezuka and F. O'Carroll and A. Hori and Y. Ishikawa. Pin-down Cache: A Virtual Memory Management Technique for Zero-copy Communication. 12th IPPS.

[11] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24 2000.

[12] 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.

[13] 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 proceeding of IPDPS, April 2004.

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

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

[16] 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.

[17] OSU Network-Based Computing Laboratory. MPI over InfiniBand Project. http://nowlab.cis.ohio-state.edu/projects/mpi-iba/index.html, January 2003.

[18] J. Worringen, A. Gaer, and F. Reker. Exploiting Transparent Remote Memory Access for Non-Contiguous and One-Sided-Communication. In *Proceedings of the 2002 Workshop on Communication Architecture for Clusters (CAC)*, April 2002.