Design Alternatives for Implementing Fence Synchronization in MPI-2 One-sided Communication for InfiniBand Clusters *

G. Santhanaraman, T. Gangadharappa, S. Narravula, A. Mamidala and D. K. Panda
Department of Computer Science and Engineering,
The Ohio State University,
{santhana, gangadha, narravul,mamidala, panda}@cse.ohio-state.edu

Abstract

Scientific computing has seen an immense growth in recent years. The Message Passing Interface (MPI) has become the de-facto standard for parallel programming model for distributed memory systems. As the system scale increases, application writers often try to increase the overlap of computation and communication. The MPI-2 standard expanded MPI to include one-sided communication semantics that has the potential for overlapping computation with communication. In this model synchronization between processes needs to be done explicitly to ensure completion before using the data. Fence is one of the mechanisms of providing such synchronization in the one-sided model.

In this paper, we study a set of different alternatives for designing the fence synchronization mechanisms. We analyze the various trade-offs of these designs on networks like InfiniBand that provide Remote Direct Memory Access (RDMA) capabilities. We propose a novel design for implementing fence synchronization that uses RDMA write with Immediate mechanism (Fence-Imm-RI) provided by InfiniBand networks. We then characterize the performance of different designs with various one-sided communication pattern microbenchmarks for both latency as well as overlap capability. The new Fence-Imm-RI scheme performs the best in scenarios that require low synchronization overhead as well as good overlap capability (close to 90% overlap for large messages) as opposed to the other designs that can provide either low synchronization overhead or good overlap capability.

1 *Introduction*

Scientific computing has seen a dramatic growth in the past decade. The demand for computing cycles in scientific simulation is growing faster than processor speed. As we advance into an era of petascale science, High End Computing (HEC) systems are continuing to meet the requirements of several grand challenge applications. At the same time there has been greater emphasis on parallel programming models as well as the middleware that supports these models in order to extract maximum performance. One of the key features desired from these programming models and middleware libraries is the ability for an application to overlap computation and communication.

One-sided communication is emerging as a promising parallel programming model especially for applications with irregular communication patterns. In this model, the data transfer operations are asynchronous in nature. The sender (origin process) alone is involved in the data transfer using a get or a put operation and there is no explicit synchronization with the receiver process for every data transfer. This model aims to improve performance by decoupling synchronization from data movement. This inherently allows the programs to be less synchronizing, the synchronization being limited only to indicate when the memory is available for reading/writing by the remote process, as well as to indicate when the data transfer has been completed from or into the remote process's memory. To reduce synchronization, the applications also communicate in cliques, near neighbor communication (e.g., PDE solvers, molecular dynamics simulations) and cartesian grids (e.g., FFT solvers) [1, 7, 2].

MPI (Message Passing interface) [10] has been the *de-facto* parallel programming model for distributed memory systems. The MPI-2 [16] standard added the one-sided communication model, also known as the Remote Memory Access (RMA) model. This kind of model provides the potential to overlap computation with communication. MPI-2 RMA supports two modes of synchronization: (i) *active:* the target process is actively involved in the synchronization and (ii) *passive:* the target process is oblivious to the on-going synchronization. The *active* mode is of two types: (i) post-start-complete-wait where only a subset of processes need to synchronize and (ii) fence in which all the processes associated with the memory window needs to synchronize.

The MPI implementations use different methods to optimize these data transfer operations and synchronizations [20, 14, 22]. Some optimizations are geared towards improving the latency of small messages. However, some of these optimizations in which data transfers are combined with synchronization operations could result in reduced scope for computation/communication overlap. In this paper, we study the dif-

*This work was supported in part U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342, #CCF-0702675 and #CCF-0833169; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, Qlogic and Sun Microsystems;.

ferent alternatives for designing and implementing an efficient *fence* method of synchronization for one-sided operations.

In the area of high performance networking, InfiniBand [13] has emerged as a popular choice and is currently powering several of the top supercomputers. The InfiniBand Architecture (IBA) provides Remote Direct Memory Access (RDMA) capability with which we can directly access the remote address space. This is a perfect semantic fit for the one-sided data transfer operations and provides low latency. However, RDMA operations pose a challenge in implementing the synchronization operations. The semantics of a fence synchronization state that all the data operations complete locally and all the issued operations complete remotely before the remote process can come out of the synchronization phase. In the context of RDMA operations, checking for remote completion pose challenges. Some of the existing fence designs circumvent this by sending completion or finish notification messages that indicate that all previous messages on this channel have been received. However this can lead to message flooding especially in large scale deployments. InfiniBand also provides RDMA write with Immediate mechanism (RI) which is an extended RDMA operation. In this paper, we leverage this mechanism to signal remote completions to implement fence synchronization.

In particular, we make the following contributions in this paper:

- Evaluate various design choices for implementing fence synchronization on modern interconnects.
- Design a novel fence synchronization that leverages InfiniBand's RDMA Write with Immediate primitives.
- Characterize the performance of the designs with several one-sided communication pattern benchmarks.

We perform a detailed performance evaluation of the different designs with various one-sided communication pattern micro-benchmarks for both latency as well as overlap capability. The new Fence-RI scheme performs the best in scenarios that require low synchronization overhead as well as good overlap capability (up to 90% overlap for large messages) as opposed to the other designs that can provide either low synchronization overhead or good overlap capability.

The rest of this paper is organized as follows: In Section 2 we provide the background for our work. The design issues are discussed in detail in Section 3. The experimental evaluations are presented in Section 4. The related work is discussed in Section 5. Conclusions and future work are presented in Section 6.

2 *Background*

In this section we briefly describe the required background for our work. In particular, we describe the necessary details of InfiniBand, MPI-2 One-sided communication, fence synchronization mechanism and MVAPICH2.

2.1 *InfiniBand*

The InfiniBand Architecture (IBA) [13] is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model). Remote Direct Memory Access (RDMA) [12] operations allow processes to access the memory of a remote node process without intervention by the remote node CPU. These operations are transparent at the remote end since they do not involve the remote CPU in the communication. IBA supports both RDMA Write and RDMA Read. These operations offer a good semantic match for MPI-2 one-sided Put and Get operations. In addition to these, InfiniBand also provides RDMA Write with Immediate operations which offers the flexibility of providing notification that the data has reached the memory in addition to directly placing the data in the remote memory. In this paper we explore using this remote notification capability to optimize the fence synchronization.

2.2 *MPI-2 One-Sided Communication*

In MPI one-sided communication (also referred to as remote memory access or RMA), the origin process (the process that issues the RMA operation) can access a target process' remote address space directly. In this model, the origin process provides all the parameters needed for accessing the memory area on the target process (also referred to as *window*) using an *MPI_Put*, *MPI_Get* or *MPI_Accumulate* operation. The completion of these operations is guaranteed by explicit synchronization calls. One-sided semantics require that these memory accesses happen within an *access epoch* and an *exposure epoch* on the origin and target process, respectively. MPI-2 semantics mandate that these data transfer operations must be used with one of the following synchronization mechanisms: (i) fence synchronization, (ii) post-start-wait-complete and (iii) lock-unlock. The first and second mechanisms are referred to as active synchronization since the target process is actively involved during synchronization. The third mechanism using lock-unlock is referred to as passive synchronization since the target process is not involved during synchronization. In this paper we focus on one-sided communication with fence synchronization mechanism.

2.3 *Fence Synchronization*

Fence is an active synchronization method which is collective over the communicator associated with the window object. Fig. 1 shows a typical fence usage scenario. The first fence call makes sure that the window on the remote process

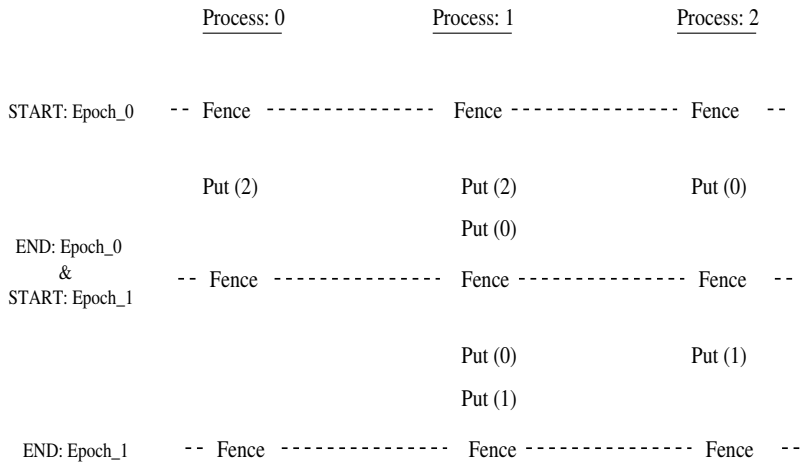

Figure 1: An Example of Fence Usage

is ready to be accessed. A process may issue one-sided operations after the first call to fence returns. The next call to fence or the second fence completes the one-sided operations issued by this process as well as the operations targeted at this process by other processes. An implementation of fence synchronization must support the following semantics: A one-sided operation cannot access a process's window until that process has called fence, and the second fence on a process cannot return until all processes needing to access that process's window have completed doing so. In addition, the second fence also needs to start the next access epoch as shown in Fig. 1. The one-sided Put operations are issued within the fence epochs. In the figure Put(x) indicates a Put operation to Process:(x).

2.4 *MVAPICH2*

MVAPICH2 is a high performance implementation of MPI-2 over InfiniBand. The implementation is based on MPICH2. As a successor of MPICH [9], MPICH2 [3] supports MPI-1 as well as MPI-2 extensions including one-sided communication. In addition MVAPICH2 supports efficient one-sided communication by taking advantage of InfiniBand RDMA Write and RDMA Read operations. MVAPICH2 is available as an open source distribution and is currently being used by more than 940 organizations worldwide including several several high end computing platforms [18].

3 *Design Alternatives*

In this section we discuss the design choices for implementing fence mechanisms, identify the limitations and propose our optimizations.

In the MPI implementations derived from MPICH2 [3, 18, 22], there are two options for implementing fence: i) Deferred and ii) Immediate. In the Deferred approach, all the operations and synchronizations are deferred till the subsequent fence. In the Immediate method, the synchronization and communication operations happen as they are issued. We explore the design issues involved in both these approaches.

As described in the previous section, a fence call needs to provide two functionalities, (i) it completes the previous epoch i.e. it ensures that all the preceding RMA operations have completed and (ii) it begins the next exposure epoch.

Next we describe the design for implementing fence using the Deferred Approach.

3.1 *Deferred Method using two-sided communication (Fence-Def)*

In this design, the first fence call does nothing and returns immediately. All the ensuing one sided operations are queued up locally. All the work is done in the second fence, where each process goes through its list of queued operations to determine its target processes. This information is stored in an array and in the second fence operation a MPI_Reduce_scatter operation is performed to let every other process know if it is the target of RMA operations from this process. The remote process can then wait for the RMA operations from these nodes. The last RMA operation from each process is conveyed to the remote process by setting a flag in that RMA message. Since the deferred approach is based on two-sided, the remote process is involved in receiving the RMA message and by looking at the flag, it ensures that it has received all the messages from that process. Since all the RMA messages are queued and issued during the fence, certain optimizations can be done that can improve the latency of the messages as well as reduce the overhead of the fence operations. However, there is no scope for providing overlap using this approach. In this design there is a notion of a remote agent that can handle incoming one-sided and synchronization messages and we refer to this two sided based design as *Fence-Def*.

3.2 *Immediate Method using RDMA Semantics*

Next we discuss fence implementations those use immediate approach and RDMA semantics of the interconnects for communication operations. This is the main focus of our work since we are interested in fence implementation on networks that support RDMA semantics.

One of the main challenges in designing fence for RDMA operations is the detection of remote completion of the Put operations.

One approach to handle remote completion is to wait for local completions and then issue a Barrier operation. This seems perfectly plausible as the Barrier is called after all the Puts are issued and completed. However this does not completely guarantee correctness as shown in Fig. 2. There is scope for the Barrier messages to overtake the Put issued to process 3 as the Barrier can be implemented in a hierarchical fashion and can complete earlier than the Put. If there is a hardware implementation of Barrier and the underlying hardware guarantees that the messages are not overtaken, only then this is a valid solution but not otherwise.

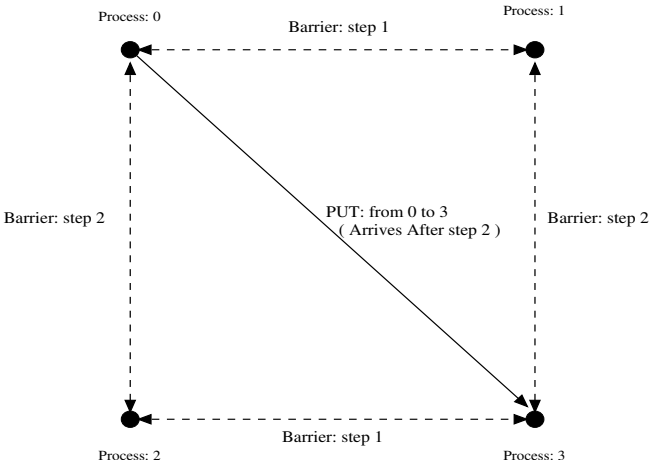

Figure 2: Barrier Messages overtaking Put

Another method of handling remote completion is by sending completion or finish notification messages that indicate that all messages on this channel have been received. There are some limitations of this approach with increasing scale.

3.2.1 *Basic Design for Fence (Fence-Imm-Naive)*

The MVAPICH2 library takes advantage of RDMA Read and Write operations to improve the performance of contiguous Get and Put operations. These one-sided operations are issued immediately. The one-sided based implementation provides higher bandwidth for large put and get messages than the two sided based design (Deferred method) and also provides greater potential for overlap of computation and communication. The current fence implementation is based on this design and is shown in Fig. 3. In order to completely implement the fence usage semantics shown earlier in Fig. 1, we need to support the following two functionalities: i) ensure local and remote completion of operations in the current epoch and ii) indicate the beginning of the next access epoch.

In this approach, polling for local completions are done to make sure that the issued one-sided operations are completed locally. For Get operations which are implemented on top of RDMA Read, local completion is sufficient to indicate that the Get operations are complete. The Put operations which are based on RDMA Write need remote completions. To handle this, a finish message is sent on each channel on which a put operation is issued to indicate that it has sent all the messages on that channel. Since the RDMA write operations on the same channel are ordered, when the finish message is received, all the RMA operations issued previously to that node are assured to be completed. Polling for local completions is done to make sure that all the messages sent have completed locally. A Reduce_scatter operation is used to let a process know if it is the target of RMA operations. The target node then waits for finish messages from all these nodes. At this point, the fence has finished completion of messages for that epoch. The next part is to indicate to all the other processes that the next epoch can begin and it is safe to access the window. The current design posts a flag to every other process to indicate that the window can now be safely accessed for the next epoch. This results in all pair-wise synchronization of the processes. This is a naive approach and leads to flood of messages in the network. We will refer to this approach as *Fence-Imm-Naive*.

This design has several drawbacks that need to be addressed. From the description of the design in the previous section, we can see that there could be two potential floods of messages during the fence. The first is a flood of finish messages to handle remote completion if the process is communicating with several peers. The second flood is the flood of messages to post a flag to indicate that the window can be accessed for the next epoch.

3.2.2 *Fence Immediate with Optimization (Fence-Imm-Opt)*

As an optimization to this approach, we use a barrier instead of the pair-wise synchronization to indicate the beginning of the next epoch. This alleviates the second flood of messages described above. Fig. 4 describes this approach and is a more scalable solution since it uses O(logn) communication steps. We refer to this approach as *Fence-Imm-Opt*.

These approaches described above still has the issue of completion messages being sent on all the channels. As the number of processes scale to large number, this could become a bottleneck. We propose a new design that uses the remote notification provided by the InfiniBand networks to design a novel and scalable fence implementation.

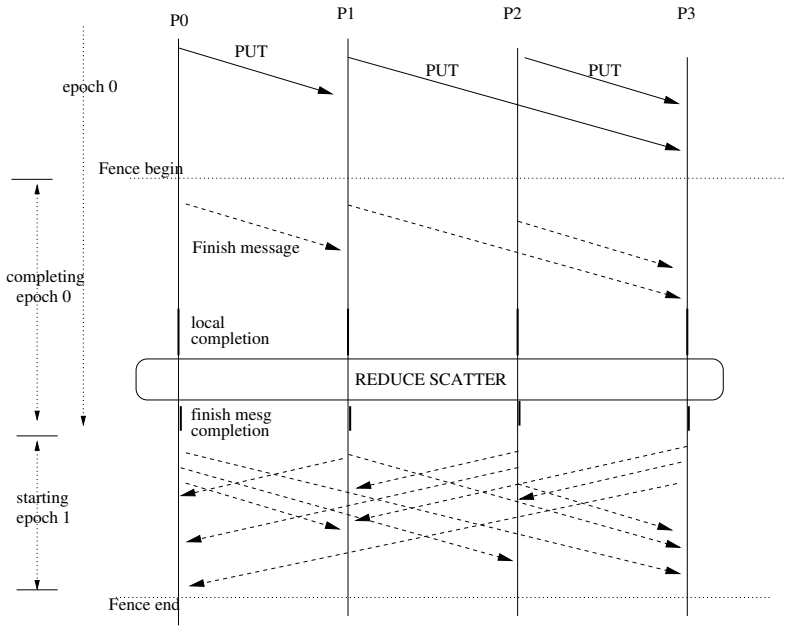

Figure 3: Fence-Imm-Naive

3.2.3 *New Scalable Fence Design With Remote Notification (Fence-Imm-RI)*

In this section we describe our new scheme which is also an Immediate method, but offers greater scalability. The new fence implementation is shown in Fig. 5. The main design and implementations issues are as follows:

Remote notification of one-sided operations: As described earlier, one approach to handle remote notifications is by flushing all the channels using a finish message. However, this approach is not scalable as it could lead to a flood of messages. In this design, we use the RDMA Write with Immediate operations to issue Put operations which creates a completion entry on the remote node. After polling for local completions, the remote node is informed of the number of such operations from all the processes through an MPI_AllReduce call. The remote node then polls till it receives completion notifications for that many number of RDMA write with Immediate operations. The completion of the Get operations is handled by waiting for local completions for the RDMA Read operations. This eliminates the first flood of messages.

Notification of beginning of next epoch: The next part is to indicate the beginning of the next epoch, i.e., to make sure that it is safe to access the window for the next epoch. It is to be noted that MPI calls provide assertions that can be used to give hints if there are no preceding or succeeding one-sided operations and in that case the fence can be optimized. Here we do not handle the assertions, but look at the general case. In our design, we use a MPI_Barrier call to indicate the beginning of the next epoch. As mentioned earlier, typical Barrier implementation uses log(n) communication steps leading to a scalable solution. One trade-off of using this approach is that it forces everyone in the group to synchronize and we might lose out on some finer grain synchronization between a subset of members of the fence group.

Preposting Receive Descriptors: One issue with using RDMA Write with immediate functionality is the need to prepost receiver descriptors. We currently handle this issue by preposting a fixed number of receive descriptors initially and repost additional descriptors in the fence synchronization call. We post additional receives on receiving RDMA write completions. However, in cases where the fence synchronization is not called often and there are extremely large number of Put operations, there is a scenario in which we might run out of receive descriptors. One solution to this approach is to use the InfiniBand Shared Receive Queue (SRQ) mechanism [21] which allows efficient sharing of receive buffers across many connections. When the number of available buffers in the shared queue drops below a low watermark threshold, an interrupt can be generated and additional buffers are posted. Another approach is to use an asynchronous thread that can post the receives.

Henceforth we will refer to this approach as *Fence-Imm-RI*. In this work we have focused on InfiniBand Architecture. However, similar designs can be proposed for other interconnects as well as modern architectures for petascale systems that can provide remote completion mechanisms for RDMA operations.

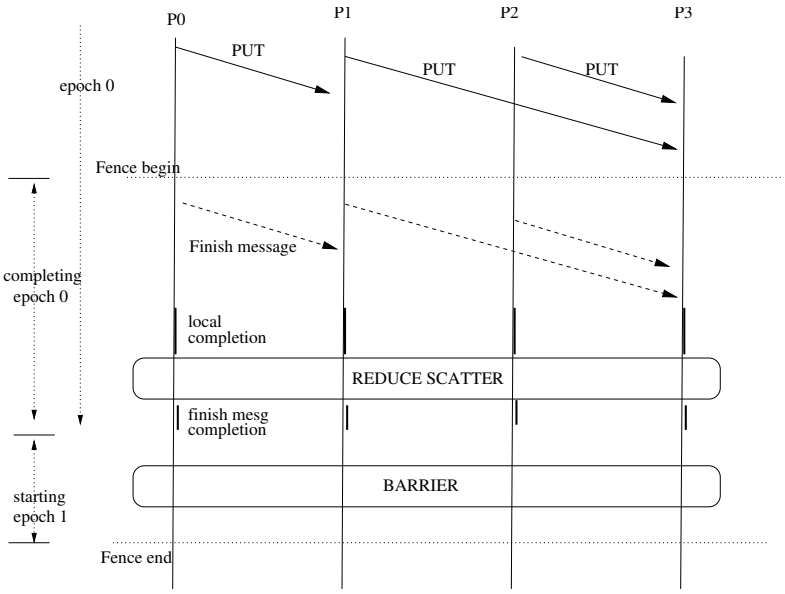

Figure 4: Optimized Design (Fence-Imm-Opt)

4 *Experimental Results*

In this section we present the experimental evaluation of the different fence designs. We characterize the performance of the proposed designs with the different micro-benchmarks representing various communication patterns.

4.1 *Experimental testbed*

Our experimental testbed is a 64 node (512-core) Intel cluster. Each node of our testbed is a dual processor (2.33 GHz quad-core) system with 4 GB main memory. The CPUs support the EM64T technology and run in 64 bit mode. The nodes support 8x PCI Express interfaces and are equipped with MT25208 HCAs with PCI Express interfaces. A Silverstorm 144 port switch is used to connect all the nodes. The operating system used is RedHat Linux AS4. All the experiments are run with one process per node configuration.

4.2 *Methodology*

In this section we describe the methodology for our evaluation. First we demonstrate the overlap capabilities of one sided based implementations as compared to one sided communication over two sided based implementations. Next, we focus on the synchronization overhead of our new Fence-Imm-RI design comparing it with implementations through a set of micro benchmarks and finally we compare the different designs for a Halo communication pattern benchmark.

4.2.1 *Overlap*

In this section we demonstrate the overlap potential for our one-sided immediate approaches compared with the two sided implementation. Each process issues Put calls to its neighbor between two fence synchronization calls. Increasing amount of computation is inserted after the Put call and before the second fence call. The overlap is measured as the amount of computation that can be inserted without affecting the overall latency. The experiment was run for varying message sizes. The results are shown in Fig. 6. We observe that the two sided Deferred implementation shows virtually no overlap. This is expected because all the Put operations are deferred and issued inside the second fence and hence there is no scope for overlap. Whereas for all the Immediate approaches using one sided implementation good overlap can be achieved for message sizes beyond 16K and close to 90% overlap for message sizes larger than 64k. In the following sections we concentrate on comparing the synchronization overhead of our new fence design (Fence-Imm-RI) as compared to all the other approaches.

4.2.2 *Basic Collectives Performance*

Since the fence designs use some of the collectives in its implementation in order to exchange the number of remote operations as well as to synchronize for the next epoch, we show the baseline performance of the collective operations: Barrier, All-Reduce and Reduce_scatter first in this section. This would help us in understanding the performance of various fence designs. Table1 shows the results for up to 64 processes, for these collectives. The All-Reduce and Reduce_scatter numbers are shown for 256 bytes message size. These collectives show good scalability with 40-50 usecs latencies on 64 processes.

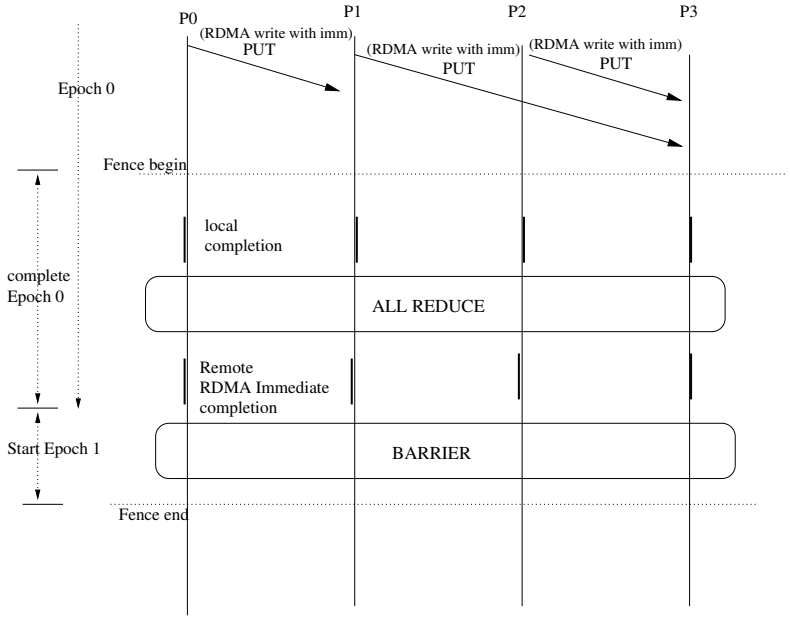

Figure 5: New design (Fence-Imm-RI)

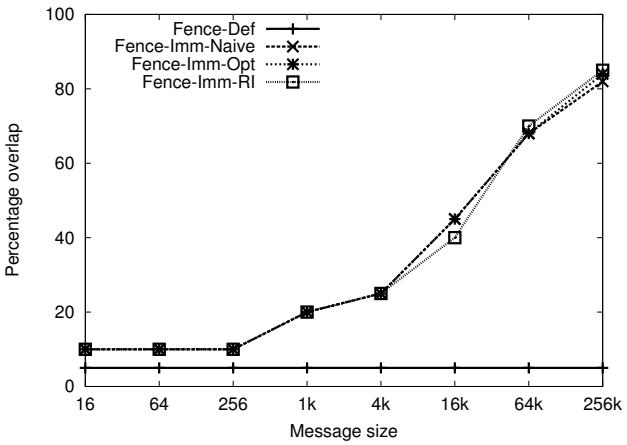

Figure 6: Overlap performance

Numprocs	Barrier	Allreduce	Reduce_Scatter
2	3.66	7.75	6.84
4	10.79	13.78	11.27
8	18.65	20.9	16.26
16	27.21	30.34	21.99
32	37.89	43.15	29.19
64	44.13	51.9	33.18

Table 1: Basic Collectives Performance (usecs)

4.2.3 *Fence Synchronization Performance*

In this section we evaluate the performance of the fence alone without any one-sided communication operations. This measures the overhead involved in a fence synchronization. The results are shown in Fig. 7. Since there are no data transfer operations, there is no overhead of the data messages in terms of local and remote completions for one-sided operations. We still need to use the collectives to inform the other processes that the fence can complete and also that the next fence epoch can begin. The Fence-Imm-Naive performs the worst, because of the all pair-wise synchronization happening to indicate the end of the epoch. The Fence-Imm-Opt and Fence-Imm-RI perform close to each other since both of them use Barrier to indicate the start of next epoch. The Fence-Imm-Opt performs slightly better than the Fence-Imm-RI, the reason for this is because the Fence-Imm-Opt uses Reduce Scatter collective as opposed to the AllReduce collective used by the Fence-Imm-RI scheme. From Table1 we can see that the Reduce Scatter collective has a lower latency than that of AllReduce. We see that the Fence-Def which uses the two sided approach performs the best, since it does not need to use additional collective to indicate the start of an epoch.

4.2.4 *Fence Synchronization with Communication Performance*

In the previous section, we evaluated the different schemes for just the fence synchronization overhead. In this section we evaluate the scalability of our fence implementations with communication operations which is more reflective of usage in a one-sided application. First we evaluate the performance of fence with a single Put of 16 bytes message size issued by all the processes. The results are shown in Fig. 8. For this pattern, we observe that Fence-Imm-Naive performs very badly. However it is interesting to compare the performance of Fence-Imm-Opt and Fence-Imm-RI. We now

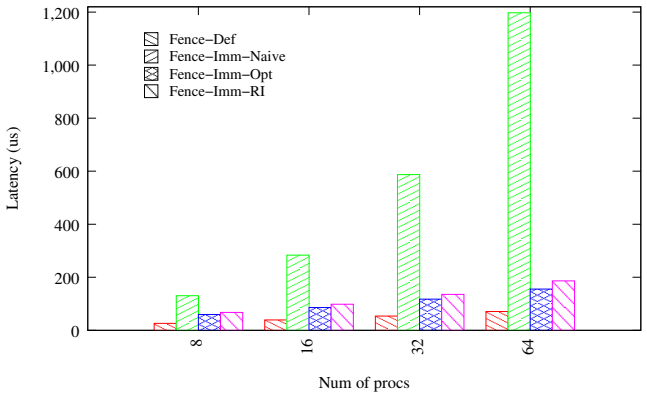

Figure 7: Fence Performance for Zero Put

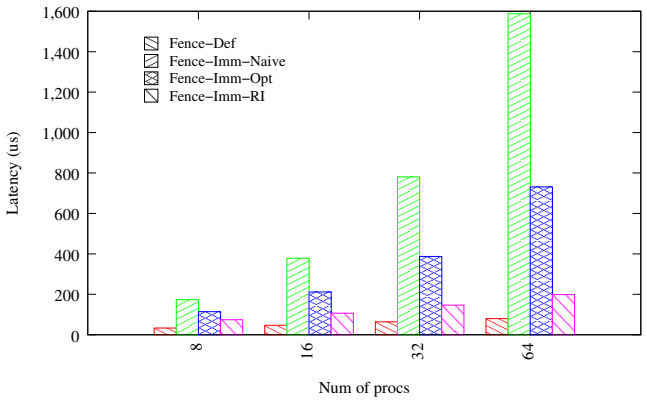

Figure 8: Fence Performance for Single Put

see that the Fence-Imm-RI outperforms the Fence-Imm-Opt scheme. The reason for this is the Fence-Imm-RI relies on the hardware RDMA-Write with immediate for remote completions, whereas the Fence-Imm-Opt has to issue completion messages which increases the overhead. This difference is magnified further in the next experiment where each process issues Puts to 8 neighbors and hence the number of completion messages increases further for the Fence-Imm-Opt. The results for this experiment is shown in Fig. 9. The two-sided approach still performs the best because it has lower overhead for small messages and can combine the data transfer and synchronization message. But it needs to be noted that it has poor overlap capability.

4.2.5 *Halo Exchange Communication Pattern*

Scientific applications often communicate in a regular pattern. Halo exchange of messages is a very popular model in which each node communicates with a fixed number (4, 8, 26, etc) of neighbors. These usually correspond to the parallel processing of multi-dimensional data in which each compute process handles a certain section of this data set. The neighbors exchange messages to handle border conditions. This communication pattern is more representative of real world applications. We simulate this halo exchange pattern for 4 and 8 neighbors and evaluate the two schemes. Every process

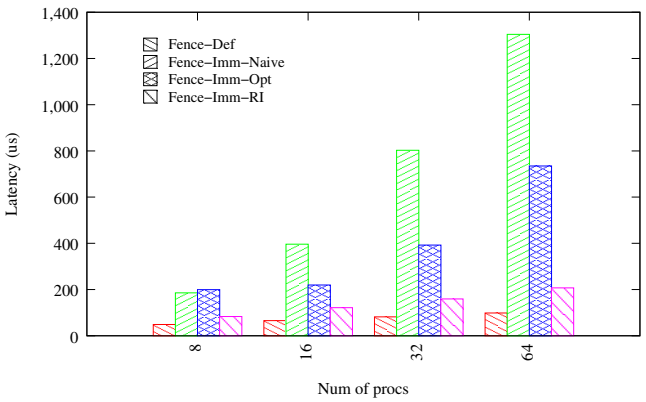

Figure 9: Fence Performance for Multiple Puts

initiates the one-sided operation with its neighbor and simultaneously performs a fixed amount of computation.

The results for 4 and 8 neighbors are shown in Fig. 10(a) and Fig. 10(b), respectively. Here we observe that our new Fence-Imm-RI scheme outperforms all the other schemes. All the immediate approaches have good computation/ communication overlap, whereas the two-sided deferred approach has very poor computation/communication overlap. The Fence-Imm-RI has reasonably low synchronization overhead and very good computation/communication overlap and hence shows the best performance.

5 *Related Work*

There are several studies regarding implementing one-sided communication in MPI-2. Some of the MPI-2 implementations that support one-sided communication are MPICH2 [3, 11], OpenMPI [4], WMPI [17], NEC [23], SUN-MPI [6].

Designs and optimizations to implement fence synchronization for MPI-2 one-sided communication using deferred approach are discussed in [22]. The NEC implementation [23] uses Allreduce and Barrier to implement fence synchronization. However they do not use RDMA Write with Immediate mechanism for remote notifications. The RDMA Write with Immediate mechanism has been explored in [15] for designing MPI_Alltoall over InfiniBand. In our work we are using this mechanism to design a scalable fence synchronization using RDMA capabilities of the network. Besides MPI, there are other programming models that use one-sided communication. ARMCI [19], GASNET [5] and BSP [8] are some examples of this model.

6 *Conclusions*

Remote completions for one-sided operations pose challenges for implementing synchronization operations. In this paper we analyzed a set of different design choices for implementing fence synchronizations on modern iterconnects. We proposed a scalable design for implementing fence synchroniza-

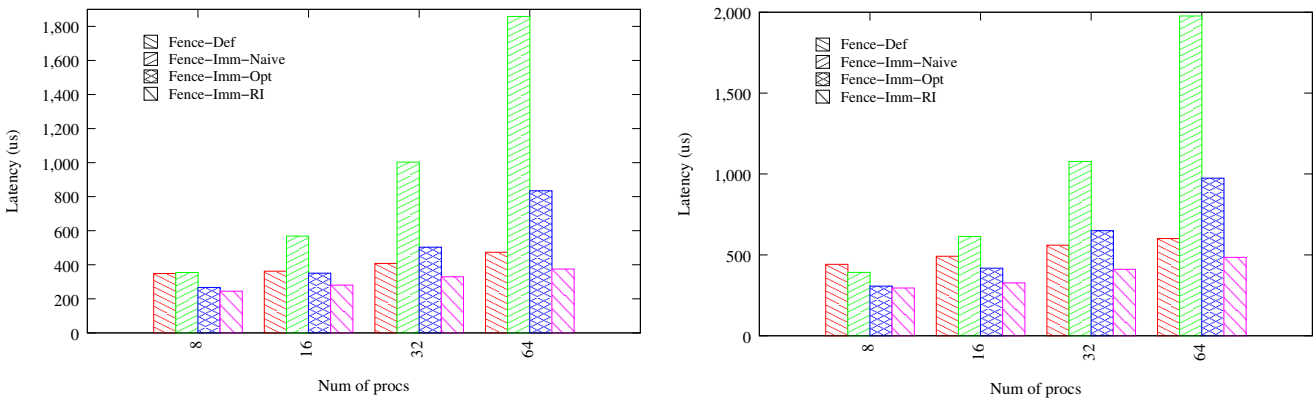

Figure 10: Fence performance with Halo Exchange: (a) 4 neighbors and (b) 8 neighbors

tion that uses RDMA write with Immediate mechanism for handling remote completions. We then characterized the performance of the design with different one-sided communication pattern micro benchmarks. Our evaluation shows significant improvement for the new Fence-Imm-RI design for the different communication patterns. Our design shows good overlap of close to 90% for large message sizes. We show good scalability for increasing number of nodes. The new Fence-Imm-RI design performs the best in scenarios that require overlap capability as well as fast synchronization.

For future work we would like to rewrite popular benchmarks to use one-sided communication with fence synchronization in an effective manner. We plan to study the impact of our designs on these benchmarks as well as on real world applications.

References

[1] GROMACS. http://www.gromacs.org/.

[2] PETSc. http://www-unix.mcs.anl.gov/petsc/.

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

[4] Brian W. Barrett, Galen M. Shipman, and Andrew Lumsdaine. Analysis of Implementation Options for MPI-2 One-Sided. In *Proceedings, Euro PVM/MPI*, Paris, France, October 2007.

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

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

[7] M. Frigo and S.G. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 2005.

[8] 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.

[9] 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.

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

[11] William D. Gropp and Rajeev Thakur. An Evaluation of Implementation Options for MPI One-Sided Communication. In *PVM/MPI*, pages 415–424, 2005.

[12] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification (Version 1.0). Technical report, RDMA Consortium, April 2003.

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

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

[15] A. Mamidala, S. Narravula, A. Vishnu, G. Santhanaraman, and D. K. Panda. On using Connection-Oriented and Connection-Less transport for Performance and Scalability of Collective and One-sided operations: Trade-offs and Impact. In *PPoPP*, 2007.

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

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

[18] Network-Based Computing Laboratory. MPI over InfiniBand Project. http://mvapich.cse.ohio-state.edu/.

[19] 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.

[20] G. Santhanaraman, S. Narravula, and D. K. Panda. Designing Passive Synchronization for MPI-2 One-Sided Communication to Maximize Overlap. In *IPDPS*, 2008.

[21] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue based Scalable MPI Design for InfiniBand Clusters. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

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

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