

# Designing Passive Synchronization for MPI-2 One-Sided Communication to Maximize Overlap \*

Gopal Santhanaraman

Sundeep Narravula

Dhabaleswar K. Panda

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

## Abstract

*Scientific computing has seen an immense growth in recent years. MPI has become the de facto standard for parallel programming model for distributed memory systems. MPI-2 standard expanded MPI to include one-sided communications. Computation and communication overlap is an important goal for one-sided applications. While the passive synchronization mechanism for MPI-2 one-sided communication allows for good overlap, the actual overlap achieved is often limited by the design of both the MPI library and the application.*

*In this paper we aim to improve the performance of MPI-2 one-sided communication. In particular, we focus on the following important aspects: (i) designing one-sided passive synchronization (Direct Passive) support using InfiniBand atomic operations to handle both exclusive as well as shared locks (ii) enhancing one-sided communication progress to provide scope for better overlap that one-sided applications can leverage. (iii) study the overlap potential of passive synchronization and its impact on applications. We demonstrate the possible benefits of our approaches for the MPI-2 SPLASH LU application benchmark. Our results show an improvement of up to 87% for a 64 process run over the existing design.*

---

\*This research is supported in part by Department of Energy's grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755, National Science Foundation's grants #CNS-0403342 and #CNS-0702675; grants from Wright Center for Innovation #WCI04-010-OSU-0; grants from Cisco Systems, Intel, Mellanox, QLogic, Sun Microsystems and Linux Networks; and equipment donations from Advanced Clustering, AMD, Appro, Intel, IBM, Mellanox, Microway, QLogic and Sun Microsystems.

## 1 Introduction

Scientific computing has seen a dramatic growth in the recent years. The demand for computing cycles in scientific simulation is growing faster than processor speed. Parallel systems with an 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) [8] 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. The MPI-1 [7] standard specifies message passing based on the send-receive model. This model is often referred to as the two-sided communication model. As an extension to MPI-1, the MPI-2 [12] standard added 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. As and when needed, synchronization between processes is done explicitly to ensure completion before using the data. MPI-2 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.

In the area of high performance networking, InfiniBand [10] 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 per-

fect semantic fit for the RMA communication model. It also provides other features like remote atomic operations that can be suitable for implementing certain one-sided primitives. These InfiniBand remote atomic operations could be leveraged to implement RDMA-based truly one-sided passive synchronization mechanisms. While it is important to design synchronization routines that provide low overhead in the MPI library as well as reduce CPU utilization to perform the synchronization, it is also critical that the design provides the maximum scope for computation and communication overlap to application writers. Latency hiding techniques are widely accepted as an important optimization in parallel programming. Application writers attempt to achieve this by overlapping communication operations with other independent computation in their applications. Such overlap of communication and computation is desirable for optimal system-wide resource usage thereby leading to overall better application performance. One way of achieving this is to use non blocking communication primitives that can initiate transfers at the earliest and then compute while this transfer is going on.

This leads to the following issues:

- How to design truly one-sided passive synchronization using Infiniband’s hardware atomic operations?
- How can we improve the overlap potential of a passive synchronization implementation?
- How much of these overlap benefits can be translated to actual performance improvement in one one-sided applications?

In this work we try to address all the above described issues and challenges. We make the following contributions in this paper: (i) Design RDMA-based truly one-sided passive synchronization support (*Direct Passive*) using InfiniBand atomic operations, (ii) Enhance one-sided communication progress to provide scope for better overlap that one-sided applications can leverage and (iii) perform an in-depth study and analyze the actual overlap benefits that our new design potentially provides to an application.

Our evaluation shows significant improvement in the overlap potential for the *direct passive* design that can be leveraged by an one-sided application. In addition to microbenchmarks we also demonstrate a significant improvement ranging between 58-87% in the performance of the MPI-2 one-sided version of SPLASH LU benchmark as compared to the existing design.

The rest of the paper is organized as follows. In Section 2, we provide the background for our work. The

design issues are discussed in Section 3. We further analyze the different aspects of our designs with respect to overlap in Section 4. 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

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 and MVAPICH2.

### 2.1 InfiniBand

The InfiniBand Architecture (IBA) [10] is an industry standard that defines a System Area Network (SAN) to design clusters offering low latency and high bandwidth. A typical IBA cluster consists of switched serial links for interconnecting both the processing nodes and the I/O nodes. IBA supports two types of communication semantics: Channel Semantics (Send-Receive communication model) and Memory Semantics (RDMA communication model). Remote Direct Memory Access (RDMA) [9] 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’s in the communication. IBA supports both RDMA Read and RDMA Write. These operations need to be initiated from and into buffers that are registered with the NIC.

**RDMA Atomic Operations:** InfiniBand provides two network level remote atomic operations, namely, *fetch\_and\_add* and *compare\_and\_swap*. The network interface card (NIC) on the remote node guarantees the atomicity of these operations. These operations act on 64-bit values. In the atomic *fetch\_and\_add* operation, the issuing process specifies the value that needs to be added and the remote address of the 64-bit location to which this value is to be added. On the other hand, in an atomic *compare\_and\_swap* operation, the issuing process specifies a ‘compare value’ and a ‘new value’. The value at the remote location is atomically compared with the ‘compare value’ specified by the issuing process. If both the values are equal, the original remote value is swapped with the new value which is also provided by the issuing process. If these values are not the same, swapping does not take place. In both the cases, the original value is returned to the issuing process.

## 2.2 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 to 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, only one side in the communication process might know 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 a 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 the origin and the target processes while passive synchronization only involves the origin process.

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

## 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 [7], MPICH2 [1] supports MPI-1 as well as MPI-2 extensions including one-sided communication. In addition MVA-

PICH2 supports RDMA-based active one-sided communication. However, currently passive communication operations are based on two-sided communication primitives. MVAPICH2 is available as an open source distribution and is currently being used by more than 610 organizations worldwide including several high end computing platforms [15].

## 3 Design and Implementation

In this section we discuss the issues and design challenges in implementing an efficient MPI-2 passive one-sided communications. In MPI-2 passive one-sided communication, the target process does not make any MPI calls to cooperate with the origin process for communication or synchronization. The synchronization is done through lock and unlock calls by the origin process on the window located on the target node. Locks are used to protect accesses to the protected target window affected by RMA calls issued between lock and unlock calls and to protect local load/store accesses to a locked local window executed between the lock and unlock call. MPI-2 passive synchronization supports locking in two modes: (i) exclusive mode and (ii) shared mode. Accesses that are protected by exclusive locks will not be concurrent at the window site with other accesses to the same window that are lock protected, i.e. only one process can have exclusive access to a window at a time. Shared lock mode allows multiple processes (readers/writers) to access the target window simultaneously. Accesses that are protected by a shared lock will not be concurrent at the window site with accesses protected by exclusive lock to the same window.

There are several different approaches for implementing passive synchronization. The passive synchronization could be implemented on top of two-sided communication. Another approach to implement passive synchronization when the memory window is not directly accessible by all the origin processes is by the use of an asynchronous agent at the target. This agent can cause progress to occur. One approach is to use a thread that periodically wakes up and checks for any pending one-sided requests. If there is underlying hardware support, then it can be exploited to provide truly one-sided passive synchronization.

There are several optimizations that are applicable to two-sided based approaches [18]. WMPI explored thread based one-sided communication and synchronization [13]. Previous work in MVAPICH2 used InfiniBand atomic operations to implement exclusive locks[11]. This design has some limitations that it considers only locking in the exclusive mode. In addition, this design does not guarantee immediate progress of

the one-sided operations which are deferred to the unlock phase. This can hurt the overlap potential. Our new design takes a step further and aims to address the above limitations while taking a similar approach of using hardware atomic operations. We use MVAPICH2 [15] as the framework for our design. In the current version of MVAPICH2, the passive synchronization support is based on two-sided communication primitives.

In this context we describe the two main aspects of our design: (i) efficient passive synchronization with support for locking in both exclusive and shared modes and (ii) enhancement of the scope for providing good overlap that the one-sided applications can potentially leverage. The following sections look at the design in further detail.

### 3.1 Efficient Passive Synchronization Support

Efficient passive synchronization support can be designed using InfiniBand’s remote atomic operations. Locking in exclusive mode can be implemented using InfiniBand’s atomic compare and swap operations. This approach does not involve the remote process, and hence is a truly one-sided mechanism for passive synchronization. However, since MPI-2 allows for both shared and exclusive modes of locking for passive synchronization, it is imperative that our design allows for shared mode locking to co-exist as well. The current MVAPICH2 design provides two-sided based shared mode locking and this can be extended to work coherently with our design based on remote atomic operations for exclusive mode locking.

For every window on a target process we maintain a 64-bit global lock state that is registered with the NIC to support remote atomic operations. This 64-bit variable can be accessed using RDMA atomic operations. This global lock state variable can be in one of 3 states: (i) unlocked, (ii) locked in exclusive mode and (iii) locked in shared mode.

This variable is by default initialized to the unlocked state during window creation. *MPI\_Comm\_size* is used to indicate this unlocked state. To obtain an exclusive lock as seen in Figure 1(a), a network based atomic *compare-and-swap* operation is done on this variable. If the *compare-and-swap* is successful, then the lock is obtained and the global state variable is set to the *process rank* of the origin process indicating that it is the current holder of the lock. During the unlock operation, this value is set back to the default value. Other processes trying to obtain a lock at the same time would fail and would keep trying till they obtain the lock once the holding process relinquishes the lock.

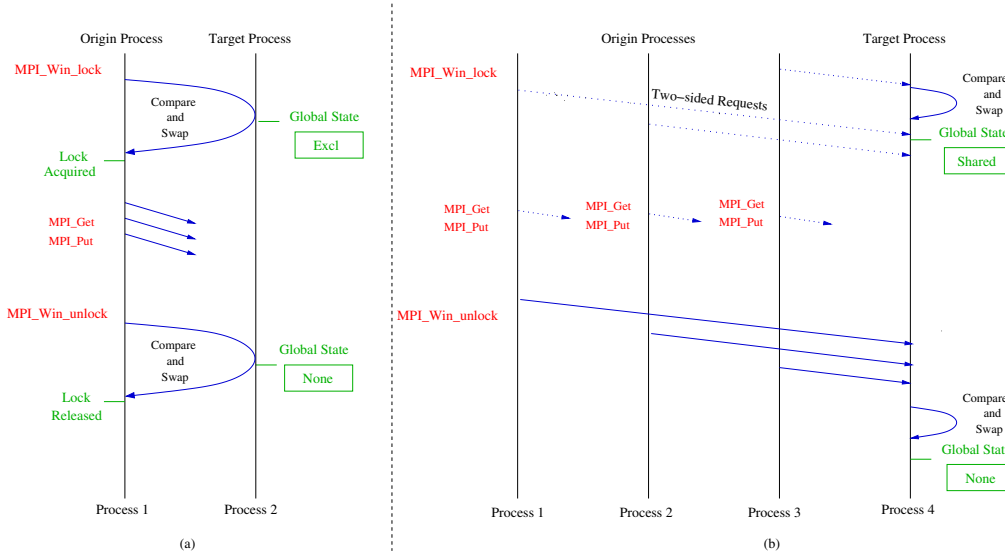
In the case of a shared lock, we use the existing two-sided approach in which a message is sent to an agent on the remote node. The agent queues up the requests and performs the issuing of lock and unlock operations locally. However, this can lead to conflicts with the exclusive mode locking and additional mechanisms are needed to handle this case.

To allow both shared and exclusive mode locking we use the following coordination mechanism. When a shared lock request is received by the remote agent it also performs an atomic *compare-and-swap* operation with the global lock state variable. If it can obtain the lock, that means there are no exclusive locks on this window, it sets the variable to a predefined value (*MPI\_Comm\_size + 1*) indicating that the lock is currently issued in shared mode and therefore all exclusive lock operations will be stalled. This agent also keeps a counter for the number of shared lock requests. When unlock operations are called, it decreases the counter variable. Once the counter variable reaches zero, it performs a *compare-and-swap* operation on global state variable resetting the global state value to the default *no lock* state. Figure 1(b) shows the basic protocol for shared mode locking. The dotted arrows in the figure indicates the operations that could be deferred to actually occur in the unlock phase when using the two-sided mechanism for obtaining shared locks.

The hardware based remote atomic operations have good scalability, but they might have the problem of flooding the network when the contention for locks is very high. However, mechanisms like exponential back off can be used to improve performance in such scenarios [11]. Since this is an orthogonal issue from the focus in this work, we have not incorporated this in our current design. We would like to incorporate this in the future.

### 3.2 Improve Overlap Scope for MPI-2 One-Sided Operations

Another aspect we aim to highlight by using the truly one-sided passive synchronization is to improve the overlap potential of the application. When two-sided approaches are used, the communication operations are often delayed to the synchronization phase and in some cases combined with an unlock synchronization call. In order to improve the progress, which leads to better overlap, we make sure that the one-sided operations within the passive epoch are issued immediately using the RDMA Write and RDMA Read InfiniBand operations. The completion of these operations are handled in the unlock operation. InfiniBand has limitations on the number of outstanding RDMA read and write operations. Hence to handle this in our design, additional



**Figure 1. Locking Mechanisms: (a) Handling Exclusive Lock and (b) Handling Shared and Exclusive Lock**

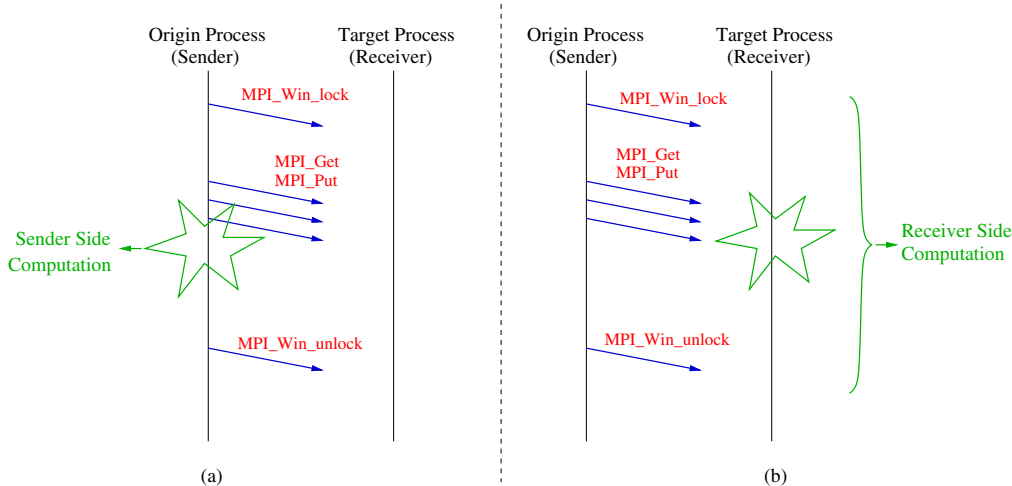
requests beyond this limit are queued up internally and issued as soon as possible.

#### 4 Overlap Analysis

In this section we analyze the different designs to understand the potential for overlap while using passive synchronization. In a passive synchronization mode overlap can be achieved at the sender side as well as the receiver side. In the sender side case, overlap can be more easily understood. Within the passive synchronization access epoch we could have computation and one-sided communication operations. If the one-sided routines are non blocking and can be initiated, then potentially we can perform computation while the initiated communication occurs in the background. More explicitly, we can do computation between MPI\_Get or MPI\_Put and the ensuing MPI\_Win\_unlock operation as long as this computation is independent or does not need the data from the one-sided operation. We refer to this as the *sender side overlap* as shown in Figure 2(a). In addition to the sender side overlap in a passive synchronization mode we can have computation on the target node while communications are occurring in its target window. This could be thought of as *receiver side overlap* as seen in Figure 2(b). An MPI-2 one-sided library geared towards maximizing overlap should provide both these kinds of overlap benefits to the application to the extent possible.

In this context, we try to analyze the two described

approaches from the overlap perspective. In the current two-sided approach there is a remote agent or receiver (in the MPI library) that handles all the one-sided communication/synchronization requests including lock, unlock, get, put. On the sender side (origin process) the lock is a local operation that is queued. The data transfers are also queued and it is only in the unlock phase that the entire lock/data transfer and unlock occurs. This kind of implementation is good when there is a requirement for lower overhead synchronization operations. Further, in this case the data transfer and the synchronization messages can be combined thus reducing the number of required network operations leading to benefits in certain scenarios. However, this results in extremely poor overlap capability for an application. Though lowering the overhead or latency of the synchronization is important, it should not come at the cost of reducing the overlap potential. Since the data transfer occurs in the unlock phase, any computation in the passive epoch cannot be overlapped at all. Also the two-sided approach requires the target node to be involved in both the computation as well as the synchronization calls. Hence this affects the on-going computation on the target node thus resulting in lower receiver overlap too. Whereas in the *direct passive* approach, the synchronizations as well as the communication operations are issued as early as possible. Further all these operations are truly one-sided because they use the underlying RDMA operations. Hence we expect better computation and communication overlap on both the sender and re-



**Figure 2. Computation and Communication Overlap: (a) Sender Side Overlap and (b) Receiver Side Overlap**

ceiver side for the *direct passive* approach.

## 5 Experimental Results

In this section we present the experimental evaluation of our *direct passive* implementation. We analyze the overlap scope with the two-sided based and *direct passive* implementations. It is to be noted that we use locks in exclusive mode for our evaluation. We then describe the results for the modified MPI-2 version of the SPLASH LU benchmark [17]. This version was obtained by modifying a shmem version of SPLASH LU benchmark to use MPI-2 one-sided calls with passive synchronization. We further profile the results of this SPLASH benchmark to analyze the performance in greater detail.

Our experimental testbed is a 64 node 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.

### 5.1 Latency

First we compare the basic performance of the two approaches: not just the cost of synchronization, but from the perspective of a data communication using passive synchronization. This is often more representative

of application behavior. We measure the time taken or latency for a lock operation followed by put and an unlock operation for various message sizes. This benchmark shows the overall latency of the two approaches

The results are shown in Figure 3. As seen in the figure, the two-sided approach performs better than the *direct passive* scheme for small messages. This is because for small messages the synchronization overhead is a significant ratio of the total time. i.e. the *direct passive* scheme needs two RDMA atomic compare and swap operations for synchronization in addition to RDMA read/write communication operation. The overhead is lower in the two-sided approach since it can combine the communication and synchronization in a single message. For larger messages, the *direct passive* scheme performs better or equally well, as the cost of data transfer is dominant. However as we have discussed in earlier sections, latency alone is not the main metric. The amount of the overlap capability the implementation can provide is critical to the performance of a one-sided application. Hence we study the designs from the overlap perspective in depth in the following section.

### 5.2 Overlap

In this section we come up with a set of micro-benchmarks that can evaluate the overlap potential both at the origin as well as the target process.

*Sender side overlap:* In this benchmark we evaluate the sender side overlap. The following is a brief description of the benchmark. Process 0 (origin process) does a lock/put/unlock on the window located on the re-

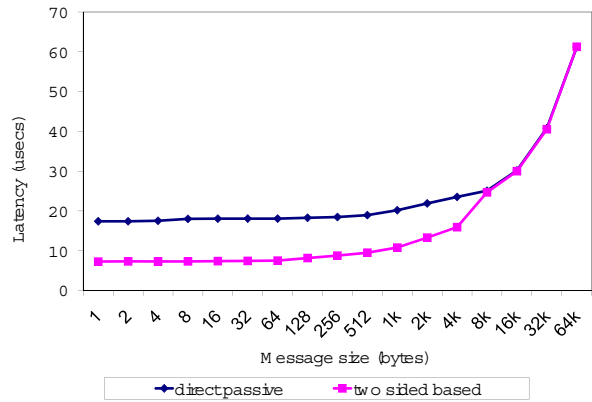
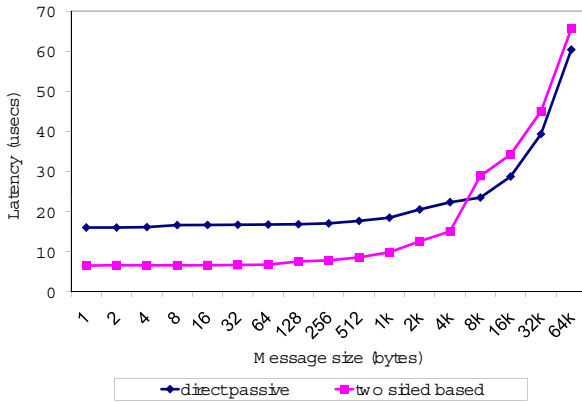


Figure 3. Basic passive performance of (a) Put and (b) Get operations

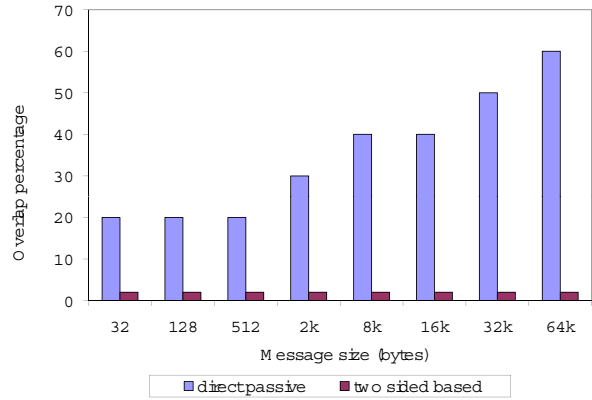
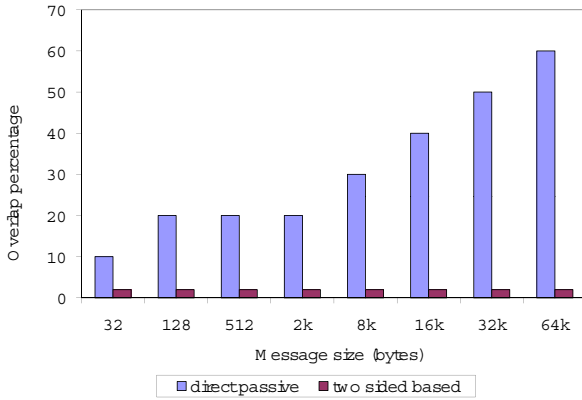


Figure 4. Overlap benefits of basic one-sided operations: (a) Put and (b) Get

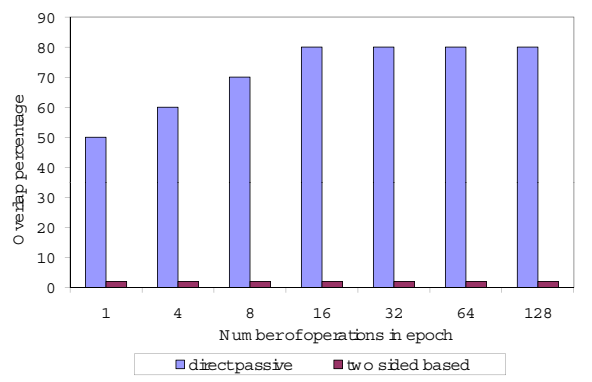
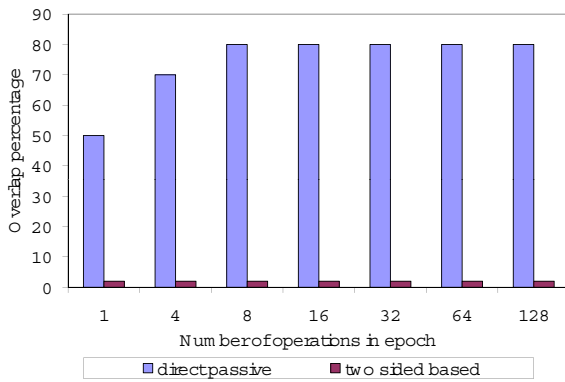


Figure 5. Overlap benefits with increasing number of operations: (a) Put and (b) Get

mote target process. The test estimates the time for the lock/put/unlock sequence. Between the get call and unlock synchronization call, increasing amounts of computation as a percentage of the estimated time are introduced. As long as the overall execution time does not change, it implies that the computation time is being absorbed or overlapped with the issued communication call.

The results for this are shown in Figure 4(a). The *direct passive* implementation shows very good overlap for large messages whereas in the two-sided approach virtually no overlap is possible because all the data transfer operations occur in the unlock phase. Please note that for the sake of visibility in the graph, we have shown a small value for the two-sided approach which can essentially be ignored. Similar results are seen for lock/get/unlock sequence shown in Figure 4(b).

*Sender side overlap with varying #operations in epoch:* This benchmark is an extension of the previous benchmark where we vary the number of get/put calls between the lock and unlock operations. The message size used is 32K. This test tries to mimic application scenarios where multiple get and put calls are issued between the synchronization operations in order to amortize the overhead of synchronization. As in the previous test increasing amounts of computation is introduced. Figure 5 shows the results of this benchmark. Once again the *direct passive* approach is able to provide much higher overlap as opposed to *no-overlap* for the two-sided approach.

*Receiver side overlap:* This benchmark tries to measure the impact of target involvement in passive mode communication on the ongoing computation. In this test there is one origin process and one target process. The test performs a fixed amount of computation on the target node. The execution time of this benchmark is the time taken by the target node to perform the fixed amount of computation. At the same time the origin process tries to access the memory window using MPI\_Get operations within a lock/unlock passive epoch. This test in effect tries to measure receiver (target) overlap, i.e., it tries to measure how much of the computation on the target node can be overlapped with the ongoing communication operations.

Figure 6(a) shows the normalized execution time of this benchmark. As compared to execution time with the *direct passive* scheme (which is normalized to 1), we observe that the two-sided approach leads to considerably higher execution times. This indicates the overhead of the target involvement for the two-sided approach or in other words this shows the reduced overlap (or lack thereof) on the target node.

*Receiver side overlap with multiple origin processes:*

We further extend the receiver overlap benchmark to multiple processes emulating one-sided application patterns. In this benchmark the overlap capability is observed in the presence of increased accesses to the target window. The message size used is 32K. Figure 6(b) shows the results in terms of normalized execution time. We see that for 64 processes, the deterioration in the execution time is about 4.5 times worse for the two-sided case as compared to *direct passive*. This is largely because of the increased communication overheads on the target node for the two-sided approach which delays the computation adversely affecting the overall execution time.

### 5.3 Application evaluation with SPLASH LU benchmark

In this section we use a modified version of the SPLASH LU benchmark to demonstrate the benefits of overlap for a one-sided application. The SPLASH LU benchmark does dense LU factorization. The dense  $n \times n$  matrix is divided into an  $N \times N$  array of  $B \times B$  blocks, such that  $n = NB$ . The blocks of the matrix are assigned to processors using a 2D scatter decomposition. The communication in LU occurs when a diagonal block is used by all the processors that require it to update the perimeter blocks they own and when the perimeter blocks are used by all processors that require them to update their interior blocks. We modified a shmem version of SPLASH LU benchmark to use MPI-2 one-sided operations. We use MPI\_Get calls to fetch the block of data and we use MPI\_Win\_Lock/MPI\_Win\_unlock passive synchronization calls. The MPI\_Win\_lock calls are used in exclusive mode.

The problem size gives the size of the overall matrix and we can vary the block size. We show the results for this benchmark for varying problem sizes and a block size of 128. This block size gave the best results.

Figure 7 shows the performance of the MPI-2 SPLASH LU benchmark for the two approaches. We observe that the *direct passive* approach always outperforms the two-sided approach. This is because the *direct passive* approach provides better overlap with reduced remote CPU involvement.

In Figure 7(a) we show the performance of SPLASH LU with a problem size 2048. We observe that the *direct passive* approach performs about 25% - 81% better than the two-sided approach. Figure 7(b) shows the performance for a larger problem size of 3000. In this case we observe higher gain ranging from 58% - 87% for the *direct passive* case as compared to the two-sided case.

In order to further understand these results, we profile the application run. In this we measure the av-

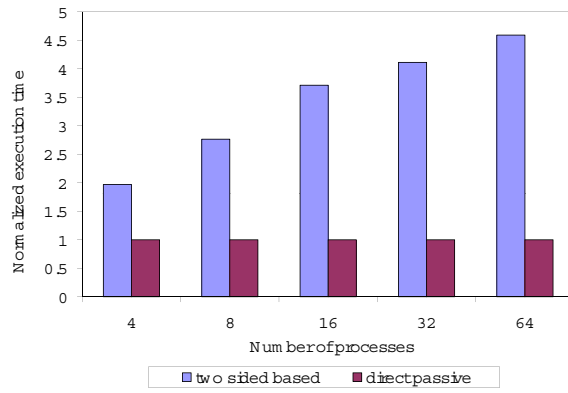
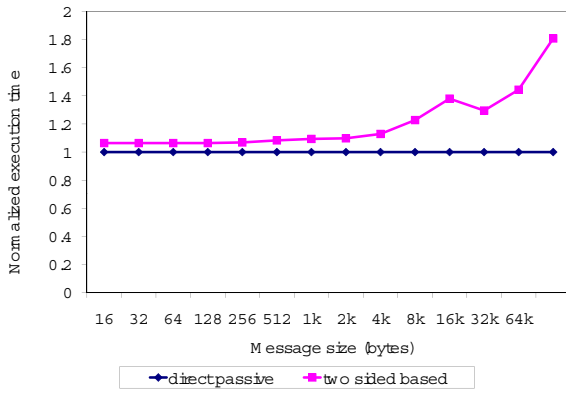


Figure 6. Receiver overlap capability with (a) two process and (b) multiple processes

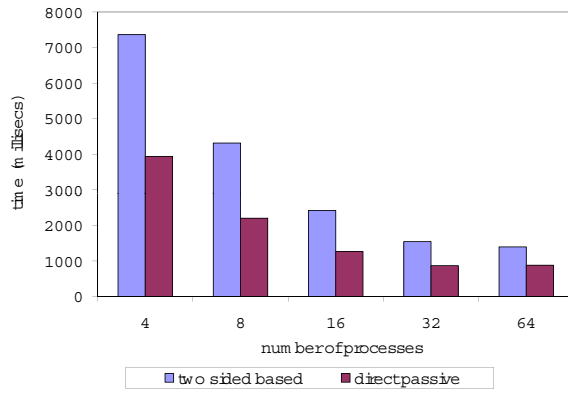
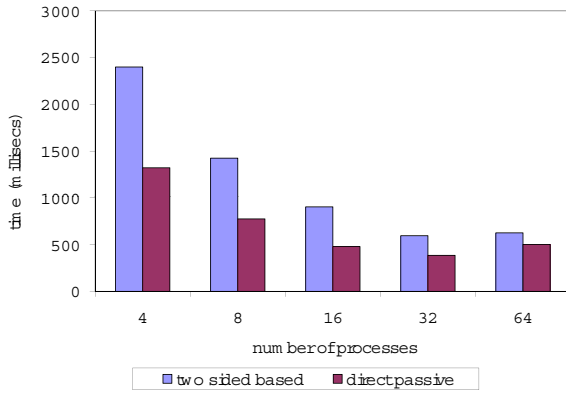


Figure 7. MPI-2 SPLASH LU benchmark: (a) Problem Size 2048 and (b) Problem Size 3000

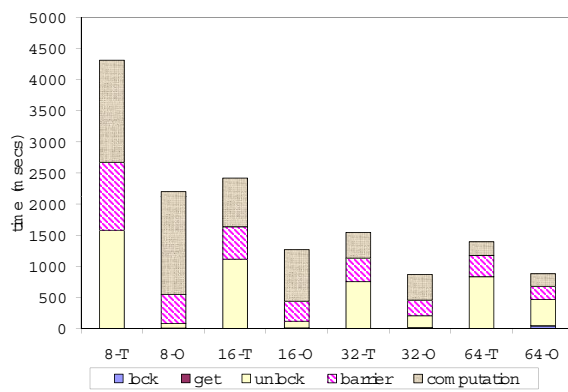
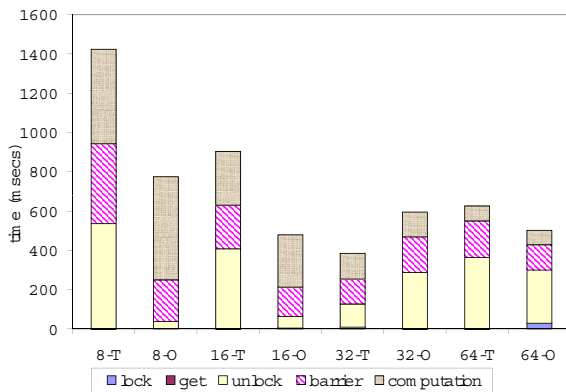


Figure 8. Timing Breakup of MPI-2 SPLASH LU: (a) Problem Size 2048 and (b) Problem Size 3000

erage time spent by the application in each of the MPI library calls. In particular, the only relevant MPI calls used in the SPLASH LU code are MPI\_Win\_lock, MPI\_Win\_unlock, MPI\_Get and MPI\_Barrier. The remaining time is classified as computation time.

In Figure 8(a) we show the timing break up of these operations for problem size 2048 for 8-64 processes. The results for problem size 3000 are shown in Figure 8(b). The legends with  $T$  stand for the two-sided based approach, and with  $O$  stand for the one-sided *direct passive* approach. As discussed in Section 4 for the two-sided approach, we observe that the lock and get operations for the two-sided approach take negligible time, since these operations are queued locally. The actual progress of these operations occurs in the unlock phase. i.e. the operations are initiated during the unlock operation. We see that the unlock operations in this case takes a large amount of time as expected. On the target node, the progress for these operations is delayed and triggered only during the MPI barrier calls. This is due to the fact that passive synchronization do not have explicit progress calls for the target node.

On the other hand, the *direct passive* scheme acquires the lock and initiates the one-sided RDMA data transfers immediately and the progress of these operations are transparent to the target node. Since this does not need the remote process to intervene, the remote process makes faster progress on its own tasks.

In addition, since the MPI Get operations does not need to wait for the target node to trigger progress, these operations move ahead faster reducing the overall application time. This aspect is clear from the numbers in Figure 8 where the two-sided approach spends a much larger time in the MPI Barrier time in performing the remote get requests which delays the computation. Consequently we also observe that the unlock time taken for the two-sided cases is significantly higher (832ms for problem size 3000 and 64 processes) as compared to the *direct passive* (421ms for problem size 3000 and 64 processes).

## 6 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 [1], WMPI [13], NEC [19], SUN-MPI [4]. Besides MPI, there are other programming models that use one-sided communication. ARMCI [16], GASNET [3] and BSP [6] are some examples of this model.

Researchers in [5] have proposed distributed queue based DLM using RDMA operations. Though this work exploits the benefits of RDMA operations for locking

services, their design can only support exclusive mode locking. Further, prior research in [14] extensively utilizes InfiniBand’s remote atomic operations for shared and exclusive mode locking, however, the main focus in their work is not in the context of MPI-2 one-sided synchronization but rather as a system-wide distributed locking service typically used in data-centers. In the context of MPI, previous work in MVAPICH2 have studied the benefits of RDMA atomic operations to efficiently implement locks in exclusive mode [11]. However their design does not take shared locks into account. OpenMPI [2] is another open source MPI implementation that supports MPI-2 standards. In OpenMPI, the library is single threaded by default and uses the two-sided approach for passive synchronization currently and depends on the target process making MPI calls to make progress. Our new design goes a step further to address the limitations of these approaches. It provides exclusive lock mode using atomic operations and shared mode locking support by extending the existing two-sided based shared locking in the MPI library and also tries to maximize the overlap potential.

## 7 Conclusions and Future Work

This paper describes our design for providing passive synchronization for MPI-2 one-sided communication with emphasis on providing overlap. We explored the use of InfiniBand remote atomic operations in our approach. In addition our design focuses on truly one-sided approaches to the extent possible. In particular, we aim to reduce the target involvement in one-sided communication using passive synchronization.

To improve the performance of MPI-2 one-sided communication, in this paper, we focused on the following important aspects: (i) *direct passive* synchronization support using InfiniBand atomic operations and (ii) enhancement of one-sided communication progress to provide scope for better overlap that one-sided applications can leverage. In addition we performed an in-depth study to characterize the sender side and receiver side overlap capabilities of our *direct passive* design.

Our evaluation shows significant improvement in the overlap potential for the *direct passive* design that can be leveraged by a one-sided application. Our micro-benchmarks show that the overlap on both the sender and receiver side is significantly enhanced using our approaches. In addition to the micro-benchmarks we also demonstrate a significant improvement ranging between 58% - 87% in the performance of an MPI-2 one-sided version of the SPLASH LU benchmark as compared to the existing design. Our detailed analysis shows that the potential benefits in this case come from the reduced re-

remote side involvement that is achievable by our design.

For future work we would like to rewrite other popular benchmarks to use one-sided communication in an effective manner. We plan to study the impact of our design on these benchmarks as well as real applications. Also the latest ConnectX InfiniBand adapters with its novel architecture have further improved the performance of the RDMA operations including atomic operations. We would like to evaluate our designs on this new emerging architecture.

## References

- [1] Argonne National Laboratory. MPICH2. <http://www-unix.mcs.anl.gov/mpi/mpich2/>.
- [2] B. W. Barrett, G. M. Shipman, and A. Lumsdaine. Analysis of Implementation Options for MPI-2 One-Sided. In *Proceedings, Euro PVM/MPI*, Paris, France, October 2007.
- [3] D. Bonachea. GASNet Specification, v1.1. Technical Report UCB/CSD-02-1207, Computer Science Division, University of California at Berkeley, October 2002.
- [4] S. Booth and F. E. Mourao. Single Sided MPI Implementations for SUN MPI. In *Supercomputing*, 2000.
- [5] A. Devulapalli and P. Wyckoff. Distributed queue based locking using advanced network features. In *ICPP*, 2005.
- [6] M. Goudreau, K. Lang, S. B. Rao, T. Suel, and T. Tsantilas. Portable and Efficient Parallel Computing Using the BSP Model. *IEEE Transactions on Computers*, pages 670–689, 1999.
- [7] 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.
- [8] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*, 2nd edition. MIT Press, Cambridge, MA, 1999.
- [9] J. Hilland, P. Culley, J. Pinkerton, and R. Recio. RDMA Protocol Verbs Specification (Version 1.0). Technical report, RDMA Consortium, April 2003.
- [10] InfiniBand Trade Association. InfiniBand Architecture Specification, Release 1.0, October 24 2000.
- [11] 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.
- [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] S. Narravula, A. Mamidala, A. Vishnu, K. Vaidyanathan, and D. K. Panda. High performance distributed lock management services using network-based remote atomic operations. CCGrid, 2007.
- [15] Network-Based Computing Laboratory. MPI over InfiniBand Project. <http://mvapich.cse.ohio-state.edu/>.
- [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] J. P. Singh, W. Weber, and A. Gupta. Splash: Stanford parallel applications for shared-memory. *SIGARCH Comput. Archit. News*, 20(1):5–44, 1992.
- [18] R. Thakur, W. Gropp, and B. Toonen. Minimizing Synchronization Overhead in the Implementation of MPI One-Sided Communication. In *EuroPVM/MPI*, September 2004.
- [19] J. Traff, H. Ritzdorf, and R. Hempel. The Implementation of MPI-2 One-Sided Communication for the NEC SX. In *Proceedings of Supercomputing*, 2000.